UNISYS
imagine it. done.

ClearPath OS 2200

The OS 2200 Internationalization Library (I18NLIB) User Manual

ClearPath OS 2200 Release 12.0

June 2009 7850 5393–006

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THIS DOCUMENT. Any product or related information described herein is only furnished pursuant and subject to the terms and conditions of a duly executed agreement to purchase or lease equipment or to license software. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, special, or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Notice to U.S. Government End Users: This is commercial computer software or hardware documentation developed at private expense. Use, reproduction, or disclosure by the Government is subject to the terms of Unisys standard commercial license for the products, and where applicable, the restricted/limited rights provisions of the contract data rights clauses.

Unisys and ClearPath are registered trademarks of Unisys Corporation in the United States and other countries.
All other brands and products referenced in this document are acknowledged to be the trademarks or registered trademarks of their respective holders.

ClearPath OS 2200

The OS 2200 Internationalization Library (I18NLIB)

User Manual

ClearPath OS 2200 Release 12.0

7850 5393–006

ClearPath OS 2200

The OS 2200 Internationalization Library (I18NLIB)

User Manual

ClearPath OS 2200 Release 12.0

7850 5393–006

Bend here, peel upwards and apply to spine.

Contents

Section 1. Introduction

1.1. Documentation Updates 1–1
1.2. About This Manual 1–1
1.3. Notation Conventions 1–2
1.4. Introduction to Internationalization (I18N) 1–3
1.4.1. Examples of Language Bias 1–3
1.4.2. Examples of Language Scripts 1–4
1.4.3. Examples of Language-Specific Rules 1–4
1.4.4. Examples of Culture-Specific Rules 1–5
1.5. Introduction to Environment Variables 1–5
1.5.1. LC_ALL 1–6
1.5.2. LC_COLLATE 1–6
1.5.3. LC_CTYPE 1–6
1.5.4. LC_MESSAGES 1–6
1.5.5. LC_MONETARY 1–6
1.5.6. LC_NUMERIC 1–6
1.5.7. LC_TIME 1–7
1.6. Introduction to Locales 1–7
1.6.1. Using Locale Names 1–7
1.6.2. Using Locale Numbers 1–8
1.6.3. Using Categories for Locale Processing 1–8
1.6.4. Using Locales and CCSs in Message Delivery 1–9
1.7. Introduction to Categories 1–12
1.7.1. Obtaining the Category Value 1–12
1.7.2. LC_ALL 1–14
1.7.3. LC_COLLATE 1–14
1.7.4. LC_CTYPE 1–16
1.7.5. LC_MESSAGES 1–16
1.7.6. LC_MONETARY 1–17
1.7.7. LC_NUMERIC 1–19
1.7.8. LC_TIME 1–19
1.8. Introduction to Coded Character Sets (CCS) 1–21
1.8.1. Identifying and Classifying Characters 1–21
1.8.2. Using I18NLIB Service Routines to Determine Character Class 1–22
1.8.3. Diagram of Interrelationship of Character Classes 1–23
1.8.4. Manipulating Characters 1–23
1.9. Introduction to the I18N Service Library (I18NLIB) 1–27
1.9.1. Components 1–27
1.9.2. Tools and Services 1–28
1.9.3. I18NLIB Service Routines 1–28
1.10. Overview of I18N Enhancements 1–28
1.11. Overview of I18N Incompatibilities 1–32

Section 2. Preparing for Internationalization

2.1. Making the Transition to the I18N Environment 2–1
2.1.1. Transition Paths 2–1
2.1.2. General Transition Tasks....2–2
2.1.3. Sample Evolutionary Transition....2–2
2.1.4. Step-by-Step Transition Procedure 2–2
2.2. Migrating Hardware to the I18N Environment 2–3
2.2.1. Unisys Terminal Considerations 2–3
2.2.2. Printer Considerations....2–3
2.2.3. Disk Storage Implications 2–4
2.3. Migrating Software to the I18N Environment 2–4
2.3.1. Migrating INFOConnect to an I18N Environment....2–6
2.3.2. Migrating IS-6000 to an I18N Environment 2–6
2.3.3. Migrating the Business Information Server to an I18N Environment....2–6
2.3.4. Migrating LINC II to an I18N Environment....2–7
2.4. Identifying the CCS of a Character When Reading Images....2–7
2.4.1. Identifying the CCS for an Image Read from a Terminal 2–7
2.4.2. Identifying the CCS for an Image Read from a System Console 2–13
2.4.3. Identifying the CCS for an Image Read from a File 2–13
2.4.3.1. SYSLIB/SLIB: Read/Write Functions for SDF Files/Elements....2–13
2.4.3.2. SYSLIB SDFI and SLIB SSDFI: Identifying the CCS for an Image Read from an SDF File/Element....2–14
2.4.4. Identifying the CCS for an Image Read from a Database....2–15
2.4.5. Identifying the CCS for an Image Read from a Network 2–17
2.5. Identifying the CCS of a Character When Writing Images....2–18
2.5.1. Identifying the CCS for an Image Written to a Terminal 2–18
2.5.2. Identifying an Image Written to a System Console....2–20
2.5.3. Identifying the CCS for an Image Written to a File 2–21
2.5.4. Identifying the CCS for an Image Written to a Database....2–23
2.5.5. Identifying the CCS for an Image Written to a Network 2–25
2.6. Using Applications in the I18N Environment....2–25
2.6.1. Modifying Application Code for the I18N Environment 2–25
2.6.2. Support for Execution Modes....2–25
2.6.2.1. TIP/HVTIP Mode Considerations....2–26
2.6.2.2. Batch Mode Considerations....2–26

2.6.2.3. Demand Mode Considerations 2–27
2.6.2.4. Real Time Mode Considerations 2–27
2.7. Testing the New Environment 2–28
2.7.1. Testing Techniques for Processing Character Data 2–28
2.7.2. Sample Verification Tests 2–28
2.8. Using MCB Transliteration Services 2–29
2.8.1. Standard Transliteration Tables 2–29
2.8.2. I18N Modifications to TIP and MCB 2–29
2.8.3. Using @MAP to Collect a Transaction for Transliteration 2–30
2.8.4. Installing the I18N Utility File 2–30
2.8.5. Using Conversion Tables 2–31
2.8.6. Creating Your Own Transliteration Tables 2–31
2.8.7. Displaying Transliteration Tables 2–32
2.9. Security Considerations 2–32
2.9.1. Security Attributes 2–32
2.9.2. TIP Security 2–33
2.10. Recovery Issues 2–33
2.11. I18NLIB Error Processing 2–34
2.12. Detecting, Displaying, and Verifying Errors 2–34

Section 3. C Compiler

3.1. C Requirements 3–1
3.1.1. Keyword Options for Accessing I18NLIB Service Routines and Enabling I18N Processing 3–1
3.1.2. Making I18N Features Compatible With Application Programs 3–1
3.1.3. Enabling I18N Features in Application Programs 3–2
3.1.4. C Compiler Calls: Sample Code for Calling the I18N Service Library 3–3
3.1.4.1. Example 1: C Compiler Call With No Options or I18N/CLIB Option 3–3
3.1.4.2. Example 2: C Compiler Call With No Options, I18N/CLIB Option, Direct Call to I18NLIB 3–4
3.1.4.3. Example 3: C Compiler Call With I18N/I18NLIB Option 3–5
3.2. C Error Structure 3–6
3.3. Error Codes 3–10
3.4. Date/Time Structures (STRFTIME and STRPTIME) 3–17
3.5. ccs_name_to_ccs_num() Function 3–17
3.6. ccs_num_to_ccs_name() Function 3–19
3.7. ccs_to_ccs() Function 3–20
3.8. close_ccs_to_ccs() Function 3–22
3.9. getenv() Function 3–25
3.10. getenvsys() Function 3–26
3.11. isalnum() Function 3–27
3.12. isalpha() Function 3–28
3.13. iscntrl() Function 3–30

3.14. isdigit() Function....3–31
3.15. isgraph() Function....3–32
3.16. islower() Function....3–33
3.17. isprint() Function....3–35
3.18. ispunct() Function....3–36
3.19. isspace() Function....3–37
3.20. isupper() Function....3–38
3.21. isxdigit() Function....3–39
3.22. locale_name_to_ccs_num() Function....3–41
3.23. locale_name_to_locale_num() Function....3–42
3.24. locale_num_to_ccs_num() Function....3–43
3.25. locale_num_to_locale_name() Function....3–45
3.26. name_and_number Packet Interface....3–46
3.27. nl_langinfo() Function....3–50
3.28. open_ccs_to_ccs() Function....3–52
3.29. putenv() Function....3–54
3.30. setlocale() Function....3–56
3.31. strfmon() Function....3–59
3.32. strftime() Function....3–63
3.33. string_collate() Function....3–67
3.34. string_transform() Function....3–69
3.35. strptime() Function....3–71
3.36. tolower() Function....3–73
3.37. toupper() Function....3–74
3.38. transform_length() Function....3–76

Section 4. COBOL Compiler

4.1. Compiler Features for Accessing I18NLIB Service Routines and Creating I18N Applications....4–1
4.2. I18N Features of COBOL Compiler Programs....4–1
4.2.1. DISP-I18N USAGE Clause....4–2
4.2.2. BYTE-I18N USAGE Clause....4–5
4.2.2.1. Creating Two Transformed Strings....4–6
4.2.2.2. Using UCOB$STRXFRM with a SORT Call....4–7
4.2.2.3. Using Transformed Key to Create Indexed Sequential File Used by PCIOS....4–9
4.2.3. SORT and MERGE Statements....4–9
4.2.4. LOCALE Clause on SELECT Statement....4–12
4.2.5. CODE TYPE Clause on SELECT Statement....4–16
4.2.6. ALPHABET Clause and PROGRAM COLLATING SEQUENCE Clause....4–17
4.3. I18NLIB COBOL Data Structures....4–18
4.3.1. Error Structure: ISL-ERR-COPY....4–18
4.3.2. Error Statuses: ISL-ERROR-STATUS....4–18
4.3.3. Date/Time Structures (STRFTIME and STRPTIME)....4–20
4.4. I18NLIB COBOL Service Routines....4–21
4.4.1. CCS_NAME_TO_CCS_NUM Service Routine....4–21
4.4.2. CCS_NUM_TO_CCS_NAME Service Routine....4–23
4.4.3. CCS_TO_CCS Service Routine....4–25

4.4.4. CLOSE_CCS Service Routine 4–27
4.4.5. CONVERT_FORM Service Routine 4–29
4.4.6. GETENV Service Routine 4–32
4.4.7. GETENVSYS Service Routine 4–34
4.4.8. ISALNUM Service Routine 4–36
4.4.9. ISALPHA Service Routine 4–39
4.4.10. ISCNTRL Service Routine 4–41
4.4.11. ISDIGIT Service Routine 4–43
4.4.12. ISGRAPH Service Routine 4–45
4.4.13. ISLOWER Service Routine 4–47
4.4.14. ISPRINT Service Routine 4–49
4.4.15. ISPUNCT Service Routine 4–52
4.4.16. ISSPACE Service Routine 4–54
4.4.17. ISUPPER Service Routine 4–56
4.4.18. ISXDIGIT Service Routine 4–58
4.4.19. LOCALE_NAME_TO_CCS_NUM Service Routine 4–60
4.4.20. LOCALE_NAME_TO_LOCALE_NUM Service Routine 4–62
4.4.21. LOCALE_NUM_TO_CCS_NUM Service Routine 4–64
4.4.22. LOCALE_NUM_TO_LOCALE_NAME Service Routine 4–66
4.4.23. NAME_AND_NUMBER Packet Interface 4–68
4.4.24. NL_LANGINFO Service Routine 4–73
4.4.25. OPEN_CCS Service Routine 4–77
4.4.26. PUTENV Service Routine 4–79
4.4.27. SETLOCALE Service Routine 4–81
4.4.28. STRFMON Service Routine 4–84
4.4.29. STRFTIME Service Routine 4–88
4.4.30. STRING_COLLATE Service Routine 4–92
4.4.31. STRING_TRANSFORM Service Routine 4–95
4.4.32. STRPTIME Service Routine 4–97
4.4.33. TOLOWER Service Routine 4–100
4.4.34. TOUPPER Service Routine 4–102
4.4.35. TRANSFORM_LENGTH Service Routine 4–104

Section 5. Basic Mode MASM

5.1. Packet for Accessing the I18NLIB Service Routines 5–1
5.1.1. Service Routine Packet: Sample Code 5–2
5.1.2. Date/Time Packet: Sample Code 5–4
5.1.3. Setting Up the Service Routine Packet 5–4
5.1.4. Generating and Accessing the Service Routine Packet 5–4
5.1.5. Generating and Accessing the Date/Time Packet 5–5
5.1.6. Transferring Control to the Service Routine 5–5
5.2. Error Structure 5–5
5.3. Error Statuses: ISL-ERR-DEFS/MSM 5–6
5.4. Date/Time Structures (STRFTIME and STRPTIME) 5–7
5.5. CCS_NAME_TO_CCS_NUM Service Routine 5–8
5.6. CCS_NUM_TO_CCS_NAME Service Routine 5–11
5.7. CCS_TO_CCS Service Routine 5–14

5.8. CLOSE_CCS_TO_CCS Service Routine 5–16
5.9. CONVERT_FORM Service Routine 5–19
5.10. GET_COLLATE_TABLE Service Routine 5–22
5.11. GETENV Service Routine.....5–26
5.12. GETENVSYS Service Routine 5–29
5.13. ISALNUM Service Routine 5–31
5.14. ISALPHA Service Routine.....5–34
5.15. ISCNTRL Service Routine.....5–37
5.16. ISDIGIT Service Routine 5–40
5.17. ISGRAPH Service Routine 5–42
5.18. ISLOWER Service Routine 5–45
5.19. ISPRINT Service Routine 5–48
5.20. ISPUNCT Service Routine 5–51
5.21. ISSPACE Service Routine.....5–53
5.22. ISUPPER Service Routine.....5–56
5.23. ISXDIGIT Service Routine.....5–59
5.24. LOCALE_NAME_TO_CCS_NUM Service Routine 5–62
5.25. LOCALE_NAME_TO_LOCALE_NUM Service Routine 5–64
5.26. LOCALE_NUM_TO_CCS_NUM Service Routine 5–67
5.27. LOCALE_NUM_TO_LOCALE_NAME Service Routine 5–69
5.28. NAME_AND_NUMBER Packet Interface 5–72
5.29. NL_LANGINFO Service Routine 5–74
5.30. OPEN_CCS_TO_CCS Service Routine 5–76
5.31. PUTENV Service Routine.....5–77
5.32. SETLOCALE Service Routine 5–80
5.33. STRFMON Service Routine 5–84
5.34. STRFTIME Service Routine 5–89
5.35. STRING_COLLATE Service Routine.....5–94
5.36. STRING_TRANSFORM Service Routine 5–98
5.37. STRPTIME Service Routine 5–105
5.38. TOLOWER Service Routine.....5–108
5.39. TOUPPER Service Routine 5–112
5.40. TRANSFORM_LENGTH Service Routine 5–115

Section 6. Extended Mode MASM

6.1. Extended Mode MASM Error Structure.....6–1
6.2. Extended Mode MASM Parameter Format 6–3
6.3. Extended Mode MASM CBEP Elements 6–4
6.4. Date/Time Structures (STRFTIME and STRPTIME).....6–5
6.5. CCS_NAME_TO_CCS_NUM Service Routine 6–5
6.6. CCS_NUM_TO_CCS_NAME Service Routine 6–8
6.7. CCS_TO_CCS Service Routine 6–11
6.8. CLOSE_CCS_TO_CCS Service Routine 6–12
6.9. CONVERT_FORM Service Routine 6–17
6.10. GET_COLLATE_TABLE Service Routine 6–20
6.11. GETENV Service Routine.....6–23
6.12. GETENVSYS Service Routine 6–26
6.13. ISALNUM Service Routine 6–29
6.14. ISALPHA Service Routine.....6–32
6.15. ISCNTRL Service Routine.....6–35

6.16. ISDIGIT Service Routine 6–38
6.17. ISGRAPH Service Routine 6–41
6.18. ISLOWER Service Routine 6–44
6.19. ISPRINT Service Routine 6–47
6.20. ISPUNCT Service Routine 6–50
6.21. ISSPACE Service Routine 6–54
6.22. ISUPPER Service Routine 6–57
6.23. ISXDIGIT Service Routine 6–60
6.24. LOCALE_NAME_TO_CCS_NUM Service Routine 6–63
6.25. LOCALE_NAME_TO_LOCALE_NUM Service Routine 6–65
6.26. LOCALE_NUM_TO_CCS_NUM Service Routine 6–68
6.27. LOCALE_NUM_TO_LOCALE_NAME Service Routine 6–70
6.28. NAME_AND_NUMBER Packet Interface 6–73
6.29. NL_LANGINFO Service Routine 6–90
6.30. OPEN_CCS_TO_CCS Service Routine 6–93
6.31. PUTENV Service Routine 6–94
6.32. SETLOCALE Service Routine 6–97
6.33. STRFMON Service Routine 6–102
6.34. STRFTIME Service Routine 6–107
6.35. STRING_COLLATE Service Routine 6–113
6.36. STRING_TRANSFORM Service Routine 6–117
6.37. STRPTIME Service Routine 6–122
6.38. TOLOWER Service Routine 6–125
6.39. TOUPPER Service Routine 6–128
6.40. TRANSFORM_LENGTH Service Routine 6–132

Section 7. Basic and Extended Mode PLUS

7.1. PLUS COPY Elements 7–1
7.2. PLUS Data Structures 7–1
7.2.1. PLUS Error Structure 7–1
7.2.2. Date/Time Structures (STRFTIME and STRPTIME) 7–3
7.2.3. Name-Number Structure 7–4
7.3. PLUS and UPLS Service Routines 7–4
7.3.1. CCS_NAME_TO_CCS_NUM Service Routine 7–4
7.3.2. CCS_NUM_TO_CCS_NAME Service Routine 7–7
7.3.3. CCS_TO_CCS Service Routine 7–10
7.3.4. CLOSE_CCS_TO_CCS Service Routine 7–13
7.3.5. CONVERT_FORM Service Routine 7–14
7.3.6. GETENV Service Routine 7–17
7.3.7. GETENVSYS Service Routine 7–19
7.3.8. ISALNUM Service Routine 7–22
7.3.9. ISALPHA Service Routine 7–24
7.3.10. ISCNTRL Service Routine 7–27
7.3.11. ISDIGIT Service Routine 7–30
7.3.12. ISGRAPH Service Routine 7–32
7.3.13. ISLOWER Service Routine 7–35
7.3.14. ISPRINT Service Routine 7–38
7.3.15. ISPUNCT Service Routine 7–40
7.3.16. ISSPACE Service Routine 7–43

7.3.17. ISUPPER Service Routine 7–46
7.3.18. ISXDIGIT Service Routine 7–48
7.3.19. LOCALE_NAME_TO_CCS_NUM Service Routine 7–51
7.3.20. LOCALE_NAME_TO_LOCALE_NUM Service Routine 7–54
7.3.21. LOCALE_NUM_TO_CCS_NUM Service Routine 7–56
7.3.22. LOCALE_NUM_TO_LOCALE_NAME Service Routine 7–58
7.3.23. NAME_AND_NUMBER Service Routine 7–61
7.3.24. NL_LANGINFO Service Routine 7–63
7.3.25. OPEN_CCS_TO_CCS Service Routine....7–66
7.3.26. PUTENV Service Routine 7–67
7.3.27. SETLOCALE Service Routine 7–70
7.3.28. STRFMON Service Routine 7–73
7.3.29. STRFTIME Service Routine....7–78
7.3.30. STRING_COLLATE Service Routine 7–83
7.3.31. STRING_TRANSFORM Service Routine....7–86
7.3.32. STRPTIME Service Routine 7–90
7.3.33. TOLOWER Service Routine 7–93
7.3.34. TOUPPER Service Routine 7–96
7.3.35. TRANSFORM_LENGTH Service Routine....7–99

Section 8. Products with I18N Features and Capabilities

8.1. Business Information Server 8–1
8.2. Communications Management System (CMS 1100)....8–2
8.3. Display Processing System 8–2
8.3.1. Changes to Existing Modules 8–3
8.3.2. Changes to Initialization Functions 8–3
8.3.3. Changes to Input Functions 8–4
8.3.4. Changes to Output Functions 8–4
8.3.5. Changes to Copy Procedures 8–5
8.4. Element Processor (ELT) 8–5
8.5. Enterprise Output Manager 8–5
8.6. Exec ASTORE Feature....8–6
8.7. Extended Language Message System (ELMS)....8–6
8.7.1. Defining Messages 8–6
8.7.2. Creating Message Module Data Banks (MDB)....8–7
8.7.3. Creating Local-Language Message Modules (MDB) 8–8
8.7.4. Testing Messages (ETEST) 8–12
8.7.5. Displaying Message Explanation Levels (EHELP)....8–13
8.7.6. Using Locales and CCSs in Message Delivery 8–14
8.7.7. Requesting Message Delivery in a Specific Language 8–14
8.7.8. Setting Environment Variables (EPROF) 8–16
8.7.9. ETEST and EHELP Processor Options....8–18
8.7.10. ELMS Error Messages....8–19
8.8. FORTRAN Compiler....8–21
8.8.1. I18NLIB Service Routines Callable from the FORTRAN Compiler....8–21

8.8.2. Using the C Compiler to Access the I18NLIB Service Routines 8–21
8.8.3. Using the COBOL Compiler to Access the I18NLIB Service Routines 8–22
8.8.4. Sample Code: FORTRAN Compiler to COBOL Compiler Entry Point 8–22
8.8.5. Error Structure and Error Statuses 8–24
8.9. General Syntax Analyzer (GSA 1100) 8–24
8.10. Interactive Processing Facility 8–24
8.10.1. Sample Code Routines 8–26
8.10.1.1. CCS Keyword Parameter (OLD Command) 8–26
8.10.1.2. $LOWERCASE System Function 8–26
8.10.1.3. $SEARCH System Function 8–26
8.10.1.4. $TRIM System Function 8–27
8.10.1.5. $UPPERCASE System Function 8–28
8.10.1.6. $CCS System Variable 8–28
8.10.1.7. $TERMINALCCS System Variable 8–29
8.10.1.8. $COLLATE System Variable 8–30
8.10.1.9. Sample Procedure Demonstrating $COLLATE 8–30
8.10.1.10. $CTYPE System Variable 8–32
8.10.1.11. Affect of $CTYPE and $COLLATE on Procedure Statements 8–32
8.10.2. Configuring, Activating, and Deactivating the I18N Feature 8–33
8.10.3. Calling Procedures with I18N Feature Active 8–33
8.10.4. Error Messages 8–33
8.11. IS-6000 Terminal Emulator 8–40
8.12. Logic and Information Network Compiler (LINC II) 8–40
8.13. Message Control Bank (MCB) 8–41
8.13.1. MCB Transliteration Service 8–41
8.13.2. General Constraints 8–41
8.13.3. List of Element Names in I18N$*UTILITY File 8–43
8.13.4. @MAP Directives 8–43
8.13.5. Sample Code for User Call 8–44
8.14. Network Database Server 8–45
8.14.1. LOCALE Clause in SCHEMA Statement 8–45
8.14.2. Sorted Chain Sets 8–45
8.14.3. FIND/FETCH Command 8–46
8.14.4. Index Sequential Record Placement 8–46
8.14.5. Schema Absolute and DMR Levels 8–46
8.14.6. C Compiler Interface 8–47
8.14.7. COBOL Compiler Interfaces 8–47
8.14.8. FORTRAN Compiler Interface 8–47
8.15. ODBC Access 8–47
8.16. Oracle Database Access 8–49
8.17. Procedure Definition Processor (PDP) 8–51
8.18. Processor Common Input/Output System (PCIOS) 8–52
8.18.1. Identifying CCSs in SSDF Files 8–52
8.18.2. Specifying Sorting Sequence for Keys in MSAM Files 8–52
8.19. Programmer's Workbench 8–53

8.20. Relational Database Server....8–53
8.20.1. Specifying a Character Set on a Column Definition....8–55
8.20.2. Specifying Collation on a Column Definition....8–56
8.20.3. Predicate Handling (WHERE Clause)....8–56
8.20.4. Handling of Implicit and Explicit CHARACTER SET and COLLATE Clauses....8–57
8.20.5. Using COLLATE Clause on DECLARE CURSOR ORDER BY Statement....8–59
8.20.6. Collation Rules for German....8–60
8.20.7. Collation Rules for Norwegian....8–61
8.20.8. Collation Rules for Spanish....8–62
8.21. Repository for ClearPath OS 2200....8–64
8.22. Service Library (SLIB)....8–64
8.23. Shared File System (SFS 2200)....8–65
8.23.1. Identifying CCSs in DSDF Files....8–65
8.23.2. Specifying Sorting Sequence for Keys in MSAM Files....8–67
8.24. Sort/Merge Utility....8–68
8.24.1. Configuration Parameters....8–68
8.24.2. Processor Parameters....8–70
8.24.2.1. DATA Processor Parameter....8–70
8.24.2.2. KEY Processor Parameter....8–71
8.24.2.3. KEYW Processor Parameter....8–72
8.24.2.4. KEY/KEYW Key Field Types....8–73
8.24.2.5. LOCALE Processor Parameter....8–73
8.24.2.6. CODE-TYPE Processor Parameter....8–74
8.24.3. Subroutine Parameters (MASM Programs Only)....8–74
8.24.3.1. KEY Subroutine Parameter....8–74
8.24.3.2. KEYW Subroutine Parameter....8–75
8.24.3.3. LOCALE Subroutine Parameter....8–76
8.24.4. Sample Runstream....8–76
8.24.5. Sample Code....8–77
8.24.6. Sorting and Collating within I18N Applications....8–78
8.24.7. Error Messages....8–79
8.25. Symbolic Stream Generator (SSG)....8–83
8.26. System Service Routine Library (SYSLIB)....8–83
8.26.1. Internationalized SYSLIB....8–83
8.26.2. SYSLIB Definitions and Routines and Their I18N Tasks....8–83

Section 9. Installing and Configuring I18NLIB Features and I18N Products

9.1. Currently Available Locales....9–1
9.2. Updating and Installing the I18NLIB Run-Time Tables....9–2
9.3. Virtual Machine for the Java Platform on ClearPath OS 2200: I18N Requirements....9–5

Appendix A. I18NLIB Error Statuses and Messages

Appendix B. List of Tables

B.1. Abbreviations for Forming Locale Names (ISO 3166:1988) B–1
B.2. Character Codes for Creating Locale Names (ISO 639:1984) B–9
B.3. Locale Names and Numbers Used With the CONVERT_FORM Service Routine B–13
B.4. Locale Names and Numbers Used With the Locale-Name-and-Number Service Routines B–13
B.5. Modified Conversion Specifications Table (STRFTIME and STRPTIME) B–14
B.6. National Replacement Character Set (NRC) B–14
B.7. Standard Transliteration Table B–15

Figures

1–1. Obtaining the Category Value 1–13
1–2. Interrelationship of Character Classes 1–23

5–1. NAME_AND_NUMBER Packet Format 5–73

Tables

3–1. STRFTIME and STRPTIME Date/Time Structures 3–17
3–2. Sequence of Arguments in a Conversion Specification 3–60
3–3. STRFTIME Conversion Specification Table 3–64

4–1. Rules for Processing Files 4–14
4–2. Sample Error Status List 4–18
4–3. STRFTIME and STRPTIME Date/Time Structures 4–20
4–4. Sequence of Arguments in a Conversion Specification 4–85
4–5. STRFTIME Conversion Specification Table 4–89

5–1. STRFTIME and STRPTIME Date/Time Structures 5–7
5–2. Sequence of Arguments in a Conversion Specification 5–85
5–3. STRFTIME Conversion Specification Table 5–90

6–1. STRFTIME and STRPTIME Date/Time Structures 6–5
6–2. Sequence of Arguments in a Conversion Specification 6–103
6–3. STRFTIME Conversion Specification Table 6–108

7–1. STRFTIME and STRPTIME Date/Time Structures 7–3
7–2. Sequence of Arguments in a Conversion Specification 7–75
7–3. STRFTIME Conversion Specification Table 7–79

8–1. ELMS Message Delivery System Error Messages 8–20
8–2. **IPF Error Messages 8–33
8–3. **EDIT Error Messages 8–36
8–4. **PROC Error Messages 8–37
8–5. **SQL Error Messages 8–37
8–6. **UA Error Messages 8–37
8–7. **VARIABLE Error Messages 8–38
8–8. **WORKFILE Error Messages 8–39
8–9. SORT Subroutines Error Messages 8–82

A–1. List of Error Statuses A–1

B–1. Abbreviations Used to Form Locale Names B–1
B–2. Transliteration Table CCS Names and Description B–15
B–3. Transliteration Table Column Names and Description B–18

Section 1
Introduction

In today's global economy, a computer system cannot compete if it is not internationalized, that is, able to handle the linguistic and cultural conventions of a variety of spoken languages. These conventions can differ across languages in the following ways:

- Characters used to write letters, numbers, and other symbols
- Language-specific characteristics such as character classification, collation, directionality, punctuation, and grammar
- Culture-specific characteristics such as date-and-time formatting, calendars, numeric and monetary formatting, measurement systems, and name-and-address conventions

The I18N Assistant is an online programming and reference guide that can be used by system administrators and applications programmers to internationalize user applications that run on the 2200 node of Unisys ClearPath OS 2200 servers.

1.1. Documentation Updates

This document contains all the information that was available at the time of publication. Changes identified after release of this document are included in problem list entry (PLE) 18647109. To obtain a copy of the PLE, contact your Unisys representative or access the current PLE from the Unisys Product Support Web site:

http://www.support.unisys.com/all/ple/18647109

Note: *If you are not logged into the Product Support site, you will be asked to do so.*

1.2. About This Manual

Note: *This manual replaces the I18NAsst Help. This is the initial version of the converted manual. Future versions will complete the conversion of the I18NAsst Help to a PDF file. During the conversion process, errors or omissions may have occurred which will be corrected in the next revision. Since this document does not currently contain an index, use the search function to locate information.*

Purpose

The purpose of this document is to

- Present general information about internationalization (I18N).

- Describe OS 2200 products that have been internationalized.
- Document the I18NLIB service routines used to internationalize OS 2200 applications and user programs.

Scope

As the companion document to the *Internationalization (I18N) Planning and Implementation Overview* (7833 4588), this document

- Supplements information about the I18N terms and concepts; provides a more detailed discussion of the I18NLIB product than what is covered in this overview; expands the descriptions of I18N features and capabilities of OS 2200 products.
- Explains how to install the I18NLIB service routines and make the transition to the I18N environment.
- Describes more fully the I18N tasks relating to migration, security, and recovery; documents the support provided for executing applications in TIP/HVTIP, batch, demand, and real-time mode.
- Includes detailed information about each I18NLIB service routine that is callable from applications and programs developed in basic and extended mode MASM, PLUS and UPLS, C Compiler, and COBOL.
- Introduces a general interface for obtaining name-and-number information for coded character sets (CCS), CCS numbers, and locale numbers. This interface is used by the Virtual Machine for the Java™ Platform on ClearPath OS 2200.

Audience

- **Primary audience:** Experienced programmers who are internationalizing applications and user programs.
- **Secondary audience:** Administrators who are responsible for installing and maintaining systems, databases, and security measures.
- **Additional audiences:** Managers and generalists who need or want I18N information at varying levels of detail.

Prerequisites

Read the companion document, *I18N Planning and Implementation Overview* for essential background and update information that is not included in this document.

1.3. Notation Conventions

Bold text indicates comments that appear in the I18NLIB sample code routines that can be copied and pasted into OS 2200 applications.

C Language Example

```
/* Is the French collation being used? */
p = getenv("LC_COLLATE");
if (p != NULL &&
   strcmp(p,"fr_FR.8859-1") = 0)
   printf("oui\o");
else
   printf("non\n");
```

MASM Example

```
. Locale category equates:

lc_all_cat    $equ  0                . LC_ALL
lc_collate_c  $equ  1                . LC_COLLATE
lc_ctype_c    $equ  2                . LC_CTYPE
```

1.4. Introduction to Internationalization (I18N)

Computer systems and applications have limited use in a worldwide market if they are developed in and for only one *natural language*, because they reflect the biases of that language.

Internationalization allows end users to develop programs and applications in their native tongue, thereby freeing them from having to know a second language. An internationalized system or application can handle various language scripts and apply the specific rules that govern a particular language or culture.

Differences addressed by I18N applications include

- Language Bias
- Language Scripts
- Language-Specific Rules
- Culture-Specific Rules

1.4.1. Examples of Language Bias

- In applications written for US English, the program messages appear in English, as do the weekdays and months.
- Dates use the month/day/year order of US English.
- Numeric and monetary formats use US English conventions.
- And most significantly, the software may only be able to handle ASCII.

1.4.2. Examples of Language Scripts

Limitations of US English Script

Applications written in US English must be modified to handle other scripts besides the plain Latin (Roman) alphabet.

- Some Latin-script languages include diacritical (ç, ñ, ô) and other letters not present in English (ø, þ, ß).
- Still other Latin-script languages do not use Ff, Jj, Qq, Vv, Ww, or Zz.

Depiction and Placement of Vowels

- Some languages use only consonants.
- If a vowel is needed, it is shown by marks above, below, or beside a consonant.
- If tone marks are used, they may affect the placement of vowels.

Ideographic Scripts

- Some languages use ideographs (symbols used to represent whole words or ideas) rather than phonetics (letters used to represent sounds).
- Other languages use both ideographs and phonetics, so an I18N application must include translators to handle both scripts.

1.4.3. Examples of Language-Specific Rules

An I18N application must be able to process natural languages based on their specific rules for grammar and punctuation, directionality, classification, and collation.

Grammar And Punctuation

- Grammar rules are different, and punctuation marks may be different or non-existent.
- The letters in some languages take different forms depending on where they appear (beginning of word, within a word, end of word, by itself).

Directionality

Characters can be written or read up, down, left to right, or right to left.

Classification

- Characters are classified as alphanumeric, alphabetic, control, digital (numeric), graphic, hexadecimal, lowercase, printable, punctuation, white space, or uppercase.
- Some languages have both uppercase and lowercase; others have only one case.

Collation

Characters are sorted and ordered differently. Two languages might use the same basic script, but sort differently because they include additional letters (ñ, þ).

1.4.4. Examples of Culture-Specific Rules

An I18N application must be able to create the proper formatting used by each natural language.

Date and Time Formatting

- Dates can be mm/dd/yy, dd/mm/yy, or a completely different format based on a different calendar.
- Times can be on 12- or 24-hour cycles or some other variation.

Calendar Formatting

Different calendars are used, and weeks on the same calendar can start on different days.

Numeric and Monetary Formatting

- Not all numbering systems use decimals.
- Some numbers are written vertically.
- Some languages use one script for letters and another for numbers.
- Monetary symbols and formatting differ for dollars, euros, pounds, rubles, lire, yen, and other currencies.

Measurement Systems

Measurement systems vary across languages and cultures, for example, metric or US equivalent.

Name And Address Conventions

There are wide variations in writing proper names and formatting such database items as addresses and telephone numbers.

1.5. Introduction to Environment Variables

Environment variables contain global data maintained by the operating system. These variables exist at both the run level and the system level.

Run-Level Environment Variables

- Set by the PUTENV service routine
- Can be retrieved by the GETENV service routine
- Contained in the user profile
- Accessible from any program or activity within a run or transaction

System-Level Environment Variables

- Set by the system administrator
- Can be retrieved by the GETENVSYS service routine
- Contained in the system profile

- Accessible from any run or transaction on the system

Processing Environment Variables

The C/UNIX/POSIX approach is used to process the environment variables in the I18N Service Library (I18NLIB). This means that the variables are defined by the Exec, stored in the run's PCT, and considered to be the default settings.

1.5.1. LC_ALL

The name of the locale to use for all I18N processing. LC_ALL contains a value that takes precedence over all other environment variables.

1.5.2. LC_COLLATE

The name of the locale to use for collation information. This information defines the relative order among collating elements in the locale. LC_COLLATE contains a value that is used by the STRING_COLLATE and STRING_TRANSFORM service routines during collation processing.

1.5.3. LC_CTYPE

The name of the locale to use for character classification and case conversion. LC_CTYPE contains a value that is used by the CONVERT_FORM service routine during character processing.

1.5.4. LC_MESSAGES

The name of the locale to use to process responses to messages. LC_MESSAGES contains values for affirmative and negative responses that are used by the NL_LANGINFO service routine during message text processing.

1.5.5. LC_MONETARY

The name of the locale to use for the rules and symbols to format monetary numeric information. LC_MONETARY contains values for currency symbols, decimal delimiters, sizes and separators for groups of digits, number of fractional digits, positioning of negative and nonnegative quantities, and other values used by the STRFMON service routine during monetary numeric processing.

1.5.6. LC_NUMERIC

The name of the locale to use for the rules and symbols to format non-monetary numeric information. LC_NUMERIC contains values for decimal delimiters and the sizes and separators for groups of digits that are used by the NL_LANGINFO service routine(s) during non-monetary numeric processing.

1.5.7. LC_TIME

The name of the locale to use for the field descriptors to format time and date information. LC_TIME contains values for names of days, weeks, months, as well as formats for time and date representations that are used by the STRFTIME and STRPTIME service routines during time-and-date processing.

1.6. Introduction to Locales

The I18N capabilities of OS 2200 software are based on the locale model of processing.

A locale:

- Defines the processing rules for the language and cultural conventions of a specific geographical region.
- Controls processing associated with identifying character attributes and collating lists.

Identifying Locales

Each locale is identified by a case-sensitive name and a number.

- The I18NLIB service routines use *category* values to identify the locale to use during processing.
- The SETLOCALE service routine sets a category value to the locale name (see 1.7).

Installing Locales

The system administrator installs locales on a system by using the Run-Time Table Builder (RTTB) processor that is released with the I18N Service Library (I18NLIB). Contact this individual if you need additional locales installed on your system or any locales removed from your system. For details about the installed locates and how to change or update the installed locates, see Section 9.

1.6.1. Using Locale Names

Predefined Locale Names

C locale

The locale model used by OS 2200 software to specify the minimal I18N environment for C-language translation.

POSIX locale

Performs the same function as the C locale, but is named differently because it is follows the conventions of a separate standards organization.

System-Defined Names

Format

```
language[_territory[.codeset]][@modifier]
```

Values

language

is the abbreviation for the natural language, such as fr (French), de (Deutsch = German), and en (U.S. English). The legal values are the two-letter language codes found in the standard ISO 639, as in en_US.ASCII.

territory

is the abbreviation for the country or locale where the language is spoken. The legal values are the two-letter country codes found in the standard ISO 3166, as in_FR, _DE, and _US.

codeset

may be appended to indicate the particular coded character set (CCS) used. For example, .8859-1 indicates ISO 8859.1, the Latin-1 or Western European CCS, as in fr_FR.8859-1.

modifier

allows additional information to be supplied for a particular usage. For example, @dict could indicate dictionary collation and @tel could indicate telephone book collation, as in de_DE.8859-1@tel. I18NLIB does not support the use of a locale modifier.

1.6.2. Using Locale Numbers

The locale number is the numeric identifier assigned by Unisys to each locale available for OS 2200 systems. The I18NLIB release tape contains a subset of these locales. For a list of the number assigned to a locale, see Section 9.

I18NLIB provides a set of name-to-number services routines in each language binding to obtain the locale number from the locale name and to obtain the locale name from the locale number. See the appropriate language section for details on these service routines.

1.6.3. Using Categories for Locale Processing

Each type of locale processing is controlled by a *category*. Each category may identify a different locale.

Currently Available Locales

The SETLOCALE service routine specifies one of the following categories to set a specific locale. For more details, refer to the SETLOCALE service routine topic.

This category	Identifies a locale to	And is used by these service routines	To perform this task
LC_TYPE	Process type, classifications, and attributes of characters.	Character classification (ISA) service routines	Specify a locale setting.
LC_COLLATE	Collate or sort lists.	STRING_TRANSFORM	Form ordering keys from character strings.
		STRING_COLLATE	Compare two character strings.
LC_ALL	Use for all categories.	SETLOCALE	Set both LC_COLLATE and LC_CTYPE to the LC_ALL value.
LC_MESSAGES	Process message text.	All	Define format and values for positive and negative responses to messages.
LC_MONETARY	Define rules and symbols for formatting monetary quantities.	STRFMON	Convert numeric monetary values to a string representation.
LC_NUMERIC	Define rules and symbols for formatting non-monetary numeric values.		
LC_TIME	Define rules and symbols for formatting date/time information.	STRFTIME; STRPTIME	Convert date/time

1.6.4. Using Locales and CCSs in Message Delivery

I18N ELMS enables OS 2200 applications to select a language by specifying a locale. A coded character set (CCS) can also be specified within a locale.

A locale name identifies the message database element (MDB) from which a message is to be retrieved. It serves the same purpose as a

- Language code in the *language_code* field of the request packet
- Language specified as the value of environment variable LANG in earlier versions of ELMS.

Selecting and Maintaining Locales and CCSs

A site or subsidiary can maintain multiple locales and CCSs. All can be stored by the Repository for ClearPath OS 2200 (formerly UREP), updated by the LOCDEF processor, and processed with the I18NLIB service routines.

A number of these locales come pre-installed with your system. You can also specify additional locales and CCSs without installing them with the LOCDEF processor.

Using Locale Sources to Obtain the Language of an Message

To obtain the language of a message to be delivered, ELMS searches locale sources in the following order:

- A language code specified in the *language_code* field of the request packet
- A locale name specified in the *locale_pref* field of the request packet
- A locale name set in the user environment variable LC_MESSAGES
- A locale name set in the system environment variable LC_MESSAGES
- A locale name set in the user environment variable LANG
- A locale name set in the system environment variable LANG
- All remaining MDB elements of a component in the order that they were registered by SOLAR

Using CCSs to Maintain Multiple MDBs

Multiple MDBs can be maintained, all having the same element name. CCS identifiers can be used to give these MDBs different version names.

Searching for Multiple Versions of an MDB

When ELMS is requested to deliver a message from a particular locale, it uses the following sequence to search for up to four different MDBs:

- CCS specified along with the locale

  You specify for component ABC the locale en_US.8859-1 and the explicit CCS CCS-A in the *CCS_pref* field of the message request packet.

  – ELMS derives an element name form the component name and locale, and then attaches the CCS to it as a version name.

  – ELMS then searches for an MDB called ABCENUS$MDB/CCS-A.

- CCS taken as part of the locale name

  You specify for component ABC the locale en_US.8859-1 and either no explicit CCS or no explicit CCS but the message is not found.

  – ELMS then searches for an implicit CCS name within the locale identifier itself.

  – The CCS is taken to be the characters that follow the period in the locale.

  – ELMS derives an element name form the component name and locale, and then attaches the implicit CCS to it as a version name.

  – ELMS then searches for an MDB called ABCENUS$MDB/8859-1.

- CCS associated with the locale name

  – ELMS checks the I18N database maintained in the Repository for ClearPath OS 2200 to determine if the locale is system-installed and if it has an associated CCS. For example, the associated CCS for locale en_US.8859-1 is "ISO8859-1".

  – If an associated CCS is found, it is appended as the version name of MDB, which ELMS then searches for the desired message. In this example, the MDB would be ABCENUS$MDB/ISO8859-1.

- CCS not specified

  – If search sequences 1, 2, and 3 fail, ELMS derives an element name from the component name and locale.

  – ELMS then searches for an MDB by that name with no version, in this case, ABCENUS$MDB.

1.7. Introduction to Categories

Categories contain locale-specific information that exists on an activity level (@XQT).

Setting Category Values

The value of a category is set by the SETLOCALE service routine. This value can be based on the setting of an environment variable.

The following categories are available with the current delivery of the I18N Service Library (I18NLIB):

- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MESSAGES
- LC_MONETARY
- LC_NUMERIC
- LC_TIME

Setting these categories affects the execution of an I18N application or program.

If a program calls a service routine that references a category value and the category value has not been set by a call to the SETLOCALE service routine, an implicit SETLOCALE service routine call occurs for the requested category.

1.7.1. Obtaining the Category Value

Figure 1–1 shows the process used by the SETLOCALE service routine to obtain a category value from the environment variables. The default locale in the following figure is

- "C" for the C language functions.
- "en_US.8859-1" if I18NLIB mode A is installed.
- "en_US.ASCII" if I18NLIB mode B is installed.

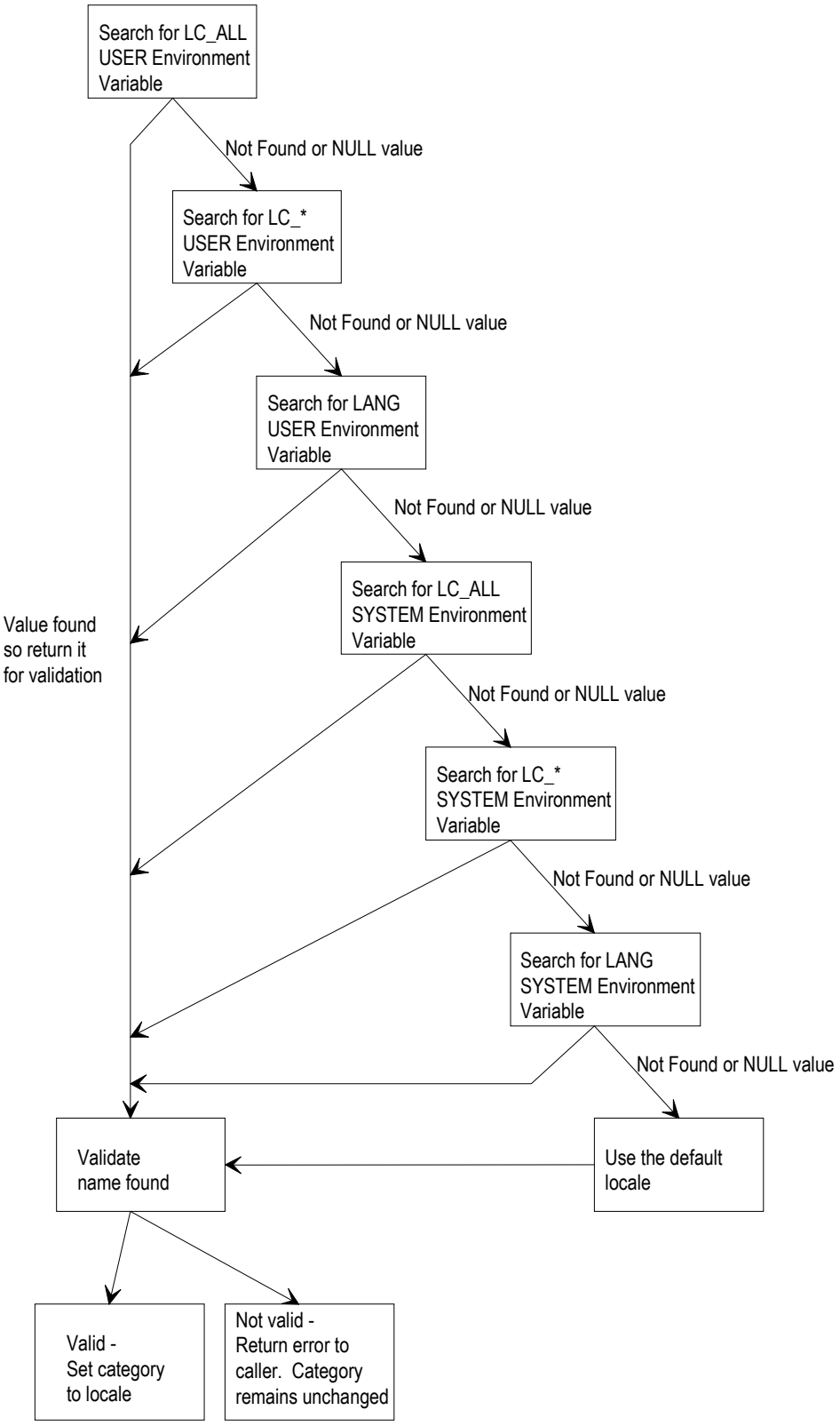


Figure 1–1. Obtaining the Category Value

1.7.2. LC_ALL

Used to set the value of all other locale categories in a program, rather than setting each one in succession. The advantage of using LC_ALL is that all error checking is done before any actions are performed. If you specify another category name, only that category is affected.

Definition Example

In this example, SETLOCALE is passed the value "_DEFAULT", which tells the service routine to set all the categories of a program to the default I18N execution environment. (If this were a C Compiler example, SETLOCALE would be passed a null string.)

```
CALL UCOB$SETLOC USING status,"LC_ALL","_DEFAULT",result
```

SETLOCALE first searches the values of all the needed environment variables to verify that they name locales supported by your system. (For a discussion of the precedence for this process, see the "_DEFAULT" locale value.)

- If the search yields a locale that is not supported, SETLOCALE returns an error status, and the locale of the program is not changed.
- If the search verifies that all environment variables name supported locales, SETLOCALE proceeds as if it had been called for each category.

1.7.3. LC_COLLATE

Affects the string-collating behavior of the STRING_COLLATE and STRING_TRANSFORM service routines.

Definition Example

In this example, the collation sequence definition is for the POSIX locale.

```
LC_COLLATE

collating-symbol <FIRST>
collating-symbol <CAPITAL>
collating-symbol <BOTH>
collating-symbol <SMALL>
collating-symbol <NONE>
collating-symbol <ACUTE>
collating-symbol <ACUTE+DOT>
collating-symbol <GRAVE>
collating-symbol <DIAERESIS>
collating-symbol <OGONEK>
collating-symbol <CIRCUMFLEX>
#
#            letter; accent; case; special
order_start forward;backward;forward;forward,position

<FIRST>
<CAPITAL>
```

```
<BOTH>
<SMALL>

<NONE>
<ACUTE>
<ACUTE+DOT>
<GRAVE>
<DIAERESIS>
   .
   .
   .
<UNDEFINED>     IGNORE;IGNORE;IGNORE
<0>
<1>    <1>
<1S>   <1>
<2>    <2>
<2S>   <2>
   .
   .
   .
<A>            <A>;<NONE>;<CAPITAL>
<a>            <A>;<NONE>;<SMALL>
<A'>           <A>;<ACUTE>;<CAPITAL>
<a'>           <A>;<ACUTE>;<SMALL>
<A!>           <A>;<GRAVE>;<CAPITAL>
<a!>           <A>;<GRAVE>;<SMALL>
<A;>           <A>;<OGONEK>;<CAPITAL>
<a;>           <A>;<OGONEK>;<SMALL>
   .
   .
   .
<ff>           <f><f>;<NONE><NONE>;<SMALL><SMALL>;<ff><ff>
   .
   .
   .
<ss>           <S><S>;<FIRST><FIRST>;<SMALL><SMALL>
   .
   .
   .
<Z/>           <Z>;<CIRCUMFLEX>;<CAPITAL>
<z/>           <Z>;<CIRCUMFLEX>;<SMALL>
<AE>           <AE>;<NONE>;<CAPITAL>
<ae>           <AE>;<NONE>;<SMALL>
   .
   .
   .
order_end
END LC_COLLATE
```

1.7.4. LC_CTYPE

Affects the behavior of the character classification (ISA) and conversion service routines.

Definition Example

In this example, the collation sequence definition is for the POSIX locale.

```
LC_CTYPE
#
# "alpha" is by default "upper" and "lower"
# "alnum" is by default "alpha" and "digit"
# "print" is by default "alnum", "punct", and the <space> character
# "graph" is by default "alnum" and "punct"
#
upper   <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
        <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
#
lower   <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
        <n>;<o>;<p>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<x>;<y>;<z>
#
digit   <zero>;<one>;<two>;<three>;<four>;\
        <five>;<six>;<seven>;<eight>;<nine>
#
space   <tab>;<newline>;<vertical-tab>;<carriage-return>;<space>
#
cntrl   <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
   .
   .
   .

# The following are examples of the Unisys extensions to the
# LC_CTYPE grammar

to-7-bit  (<a:>,<7a:>);(<u:>,<7u:>)

to-8-bit  (<7a:>,<a:>);(<7u:>,<u:>)

END LC_CTYPE
```

1.7.5. LC_MESSAGES

Defines format and values for affirmative and negative responses to message text. The operands are strings or regular expressions.

Definition Example

This example illustrates the en_US locale section for LC_MESSAGES (en_US is the locale name for English as spoken in the US).

```
LC_MESSAGES
yesexpr     "<<(><y><Y><s><S><)/>><.><*>"
noexpr      "<<(><n><N><)/>><.><*>"
END LC_MESSAGES
```

yesexpr

Defines the affirmative response to a question expecting an affirmative or a negative response.

noexpr

Defines the negative response to a question expecting an affirmative or a negative response.

1.7.6. LC_MONETARY

Defines rules and symbols used to format monetary numeric information used by the STRFMON service routine. The operands contain either formatted strings or integers.

Definition Example

This example illustrates the handling of formatted monetary quantities in the en_DK locale section for LC_MONETARY (en_DK is the locale name for English as spoken in Denmark).

```
LC_MONETARY
int_curr_symbol      "<S><D><R><SP>"
currency_symbol      "<Cu>"
mon_decimal_point    "<,>"
mon_thousands_sep    "<.>"
mon_grouping         3;3
positive_sign        " "
negative_sign        "<->"
int_frac_digits      2
frac_digits          2
p_cs_precedes        1
p_sep_by_space       0
n_cs_precedes        1
n_sep_by_space       0
p_sign_posn          1
n_sign_posn          1
END LC_MONETARY
```

int_curr_symbol

A string containing the international currency symbol.

currency_symbol

A string containing the local currency symbol.

mon_decimal_point

A string used as the decimal delimiter.

mon_thousands_sep

A string used as the separator for groups of digits to the left of the decimal delimiter.

mon_grouping

A sequence of integers defining the size of each group of digits preceding the decimal delimiter.

positive_sign

A string indicating a nonnegative value.

negative_sign

A string indicating a negative value.

int_frac_digits

An integer representing the number of fractional digits to be written in the quantity using *int_curr_symbol*.

frac_digits

An integer representing the number of fractional digits to be written in the quantity using *currency_symbol*.

p_cs_precedes

An integer indicating if a currency symbol precedes or succeeds the value of a nonnegative quantity.

p_sep_by_space

An integer indicating whether or not spaces separate the currency symbol from the value of a nonnegative quantity.

n_cs_precedes

An integer indicating if a currency symbol precedes or succeeds the value of a negative quantity.

n_sep_by_space

An integer indicating whether or not spaces separate the currency symbol from the value of a negative quantity.

p_sign_posn

An integer indicating the positioning of *positive-sign* for a nonnegative quantity.

n_sign_posn

An integer indicating the positioning of *negative-sign* for a negative quantity.

1.7.7. LC_NUMERIC

Defines rules and symbols used to format nonmonetary numeric information. The operands contain either formatted strings or integers.

Definition Example

This example illustrates the en_DK locale section for LC_NUMERIC (en_DK is the locale name for English as spoken in Denmark).

```
LC_NUMERIC
decimal_point        "<,>"
thousands_sep        "<.>"
grouping             3;3
END LC_NUMERIC
```

decimal_point

Defines the string to use as the decimal delimiter.

thousands_sep

Defines the string to use as the separator for groups of digits to the left of the decimal delimiter.

grouping

Defines the size of each group of digits preceding the decimal delimiter.

1.7.8. LC_TIME

Defines the interpretation of field descriptors to use for formatting time and date information. The STRFTIME and STRPTIME service routine reference this category value.

Definition Example

This example illustrates the POSIX locale section for LC_TIME. (The POSIX locale specifies the minimal environment for C-language translation.)

```
LC_TIME
# Abbreviated weekday names (%a)
abday   "<S><u><n>";"<M><o><n>";"<T><u><e>";"<W><e><d>";\
        "<T><h><u>";"<F><r><i>";"<S><a><t>"
#
# Full weekday names (%A)
day     "<S><u><n><d><a><y>";"<M><o><n><d><a><y>";\
        "<T><u><e><s><d><a><y>";"<W><e><d><n><e><s><d><a><y>";\
        "<T><h><u><r><s><d><a><y>";"<F><r><i><d><a><y>";\
        "<S><a><t><u><r><d><a><y>"
#
# Abbreviated month names (%b)
abmon   "<J><a><n>";"<F><e><b>";"<M><a><r>";\
        "<A><p><r>";"<M><a><y>";"<J><u><n>";\
        "<J><u><l>";"<A><u><g>";"<S><e><p>";\
        "<O><c><t>";"<N><o><v>";"<D><e><c>"
#
# Full month names (%B)
mon     "<J><a><n><u><a><r><y>";"<F><e><b><r><u><a><r><y>";\
        "<M><a><r><c><h>";"<A><p><r><i><l>";\
        "<M><a><y>";"<J><u><n><e>";\
        "<J><u><l><y>";"<A><u><g><u><s><t>";\
        "<S><e><p><t><e><m><b><e><r>";"<O><c><t><o><b><e><r>";\
        "<N><o><v><e><m><b><e><r>";"<D><e><c><e><m><b><e><r>"
#
# Equivalent of AM/PM (%p)      "AM"/"PM"
am_pm   "<A><M>";"<P><M>"
#
# Appropriate Date/Time representation (%c)
#       "%a %b %e %H:%M:%S %Y"
d_t_fmt "<percent-sign><a><space><percent-sign><b><space>\
<percent-sign><e><space><percent-sign><H><colon><percent-sign><M>\
<colon><percent-sign><S><space><percent-sign><Y>"
#
# Appropriate date representation (%x)   "%m/%d/%y"
d_fmt   "<percent-sign><m><slash><percent-sign><d><slash>\
<percent-sign><y>"
#
# Appropriate time representation (%X)   "%H:%M:%S"
t_fmt   "<percent-sign><H><colon><percent-sign><M><colon>\
<percent-sign><S>"
#
# Appropriate 12 h time representation (%X)   "%I:%M:%S %p"
t_fmt_ampm "<percent-sign><I><colon><percent-sign><M><colon>\
<percent-sign><S><space><percent-sign><p>"
#
END LC_TIME
```

1.8. Introduction to Coded Character Sets (CCS)

A CCS is a mapping of characters onto a range of numeric values that can be used to store text in files or pass text through communications paths.

A CCS includes the characters used by one or more countries, or in one or more languages. Each graphic image has an associated numeric value.

This mapping of characters enables a computer system to store character data. Standardized CCSs allow computer systems from different manufacturers to exchange this data.

Identifying CCSs

Each CCS is identified by three identification schemes.

- Unisys CCS number – Identifies in numerical sequence the CCSs provided by Unisys.
- SDF CCS number – Identifies the CCSs that can be named in an SDF control word.
- ISO number – Identifies character sets that have been standardized by the International Standards Organization (ISO).

See Appendix B for a list of the CCS and the associated CCS numbers.

Using CCSs to Enhance OS 2200 Software

By using CCSs, OS 2200 software can

- Represent a wide assortment of scripts.
- Handle 7-bit and 8-bit CCSs simultaneously.
- Use 8-bit character sets from the ISO 8859 suite, which includes characters with diacritical marks.

Without using CCSs, the software is limited to 6-bit (Fieldata) and 7-bit (ASCII) character sets. This presents certain constraints. For example, Fieldata does not allow lowercase.

1.8.1. Identifying and Classifying Characters

Sometimes natural language characters need to be identified before they can be processed. This includes

- Determining their character class
- Identifying the coded character set (CCS) to which a character belongs when reading and writing images

Classifying a Character

Classifying a character means describing the group or groups it belongs to. For example, is it alphabetic or numeric? Is it uppercase or lowercase? Is it printable? Is it a control or white space character? Some characters belong to more than one character class.

For a list of I18NLIB service routines used to determine character class, see 1.8.2.

For a diagram of character class interrelationship, see1.8.3.

Identifying the CCS of a Character

There are a number of products, features, and methods that can be used to identify the CCS of a character when reading or writing images, depending on whether the source of the character is a terminal, system console, file, database, or network.

1.8.2. Using I18NLIB Service Routines to Determine Character Class

This service routine	Identifies a character as
ISALNUM	Alphanumeric
ISALPHA	Alphabetic
ISCNTRL	Control
ISDIGIT	Digital (numeric)
ISGRAPH	Graphic
ISLOWER	Lowercase
ISPRINT	Printable
ISPUNCT	Punctuation
ISSPACE	White space
ISUPPER	Uppercase
ISXDIGIT	Hexadecimal

List of Character Class Service Routines

The character class service routines are: ISALNUM, ISALPHA, ISCNTRL, ISDIGIT, ISGRAPH, ISLOWER, ISPRINT, ISPUNC, ISSPACE, ISUPPER, and ISXDIGIT.

Character Class Service Routines: Names and Correct Character Classes

For this service routine	The correct character class is
ISALNUM	Alphanumeric
ISALPHA	Alphabetic
ISCNTRL	Control
ISDIGIT	Decimal digit
ISGRAPH	Graphic
ISLOWER	Lowercase
ISPRINT	Printable

For this service routine	The correct character class is
ISPUNCT	Punctuation
ISSPACE	White space
ISUPPER	Uppercase
ISXDIGIT	Hexadecimal digit

1.8.3. Diagram of Interrelationship of Character Classes

This diagram shows the interrelationships between character classes. For example, the letter **a** is lowercase, alphabetic, and printable.

The pattern may vary slightly from one locale definition to another.

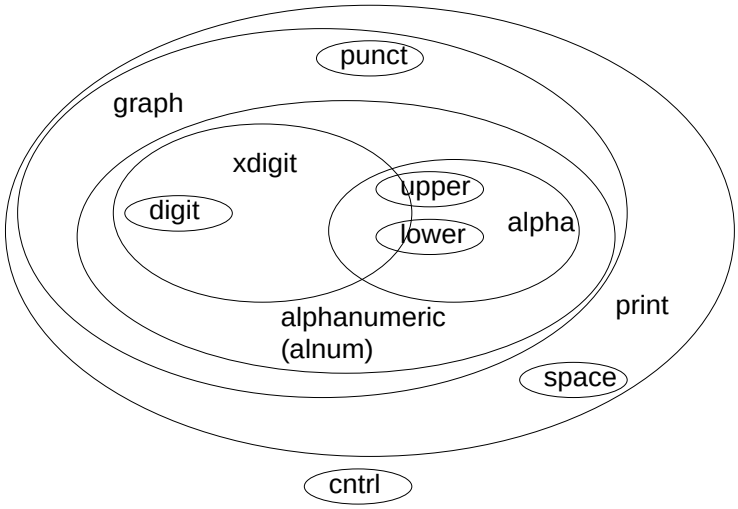


Figure 1–2. Interrelationship of Character Classes

1.8.4. Manipulating Characters

Mapping Characters from One CCS to Another

Unisys provides string transliteration tables to map characters from one CCS to another. For a list of CCSs available for transliteration, go to the MCB transliteration tables.

Transliterating 1- and 2-Byte CCSs

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS support transliterations involving both 1-and 2-byte CCSs. (Support for 2-byte CCSs is also known as multi-octet support.)

CCSs Supported

The 2-byte CCSs currently supported are

- The UCS-2 representation of ISO 10646 (also known as Unicode)
- The OS 2200 representation of kanji (LETS-J)

Possible Transliterations

In all cases, these transliterations are supported in both directions.

- 1-byte CCS to 1-byte CCS
- 1-byte CCS to 2-byte UCS-2
- 2-byte to 2-byte between UCS-2 and LETS-J

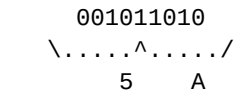
Background

To make it easier to transport characters between heterogeneous systems, it is assumed that

- The 2-byte characters are represented by using only the lower eight bits of each 9-bit byte.
- The high-order bit of the OS 2200 9-bit byte is set to 0.

Example

A Latin uppercase S with an acute accent (′) in Unicode is defined as hexadecimal 015A (/x01/x5A). The bit representation of this character on an OS 2200 system is



Converting Undefined Characters to Replacement Characters

If a character has no conversion defined, it will be converted to the replacement character specified in the CCS_TO_CCS conversion source element. The default character is '_'.

Translating Private Use Characters from One CCS to Another

The CCS_TO_CCS service routine can be modified to translate private use (user-defined) characters. The translation of characters from one CCS to another is performed by way of the UCS-2 CCS, which is the bi-polar basic multilingual plane (BMP) of the ISO-10646l character set and the basic Unicode standard. All characters are represented in hexadecimal notation.

The Unicode standard defines a private use area for private use or user-defined characters. This area is defined as the character encodings from U+E000 through U+F8FF, and is divided into two subareas:

- The corporate use subarea, which begins with encoding U+F8FF and extends downward
- The end user subarea, which begins with U+E000 and extends upward

Format Guidelines

No definite boundary exists between the corporate use and the end user subareas, and Unisys has not yet designated any part of the area for its corporate use. It is therefore recommended that all user-defined characters be mapped into the end user area, beginning with hexadecimal E000.

- The CHARMAPs have element names of *xxxx*/CHARMAP where *xxxx* is the CCS name. This element maps a symbolic representation of each character in the CCS to the encoding of the character in the CCS. All symbolic representations of characters within a CCS must be unique and must be enclosed within the angle brackets < >.
- The symbolic representation of the private use characters should be <U*xxxx*> where *xxxx* is the hexadecimal encoding of the character in the UCS2 CCS. Using this representation should avoid conflicts with existing and future character definitions in the CCS.

Restrictions

- Private use characters can only be defined into CCSs that are not completely allocated or defined by an external standard, unless the site wants to locally override the definition of a character.
- Private use characters can be added to a CCS and to the CCS-to-CCS translation, but I18NLIB does not permit the definition of a user-defined locale or CCS.

Procedure for Defining Private Use Characters

To enable translation of user-defined characters by the CCS-TO-CCS service routine, you need to first define a character in a CCS, then define the mapping of the CCS character to the UCS-2 character, and finally, rebuild and update the run-time tables.

The updates to the CHARMAP and the CCS-TO-CCS sources must be placed in the SYSLIB*I18NLIB-LANG file or in another file that contains all the sources in SYSLIB*I18NLIB-LANG plus the modified sources. The use of another file is recommended so that updates are easier for moving up levels of I18NLIB.

The procedure for defining private use characters is as follows:

1. Define the character in the CHARMAP of the CCS.

   The CHARMAP sources for all supported and some unsupported CCSs are located in the file SYSLIB*I18NLIB-LANG, which is created when I18NLIB is installed. The symbol representation of the character must be followed with the encoding of the character in the CCS. For example:

   ```
   <UE000>     /xBC/x00
   ```

   or

   ```
   <UE000>     /xFC
   ```

   where the first definition for the character <UE000> is in a 2-byte CCS with a BC encoding for the first byte and a 00 encoding for the second byte; the second definition of the character UE000 in a 1-byte CCS with <UE000> has an FC encoding. The slash (/) is the escape character and is assumed to be the escape character defined in the CHARMAP. If a different character is defined as the escape character, it must be substituted for /.

2. Define the mapping between the character in the CCS and the character in the UCS-2 CCS.

   This mapping is defined in the element SYSLIB*I18NLIB-LANG.*xxxx*/CCS-TO-CCS, where *xxxx* is the name of the CCS. This mapping is accomplished by adding to the CCS-TO-CCS source. For example:

   ```
   (<UE000>,<UE000>);
   ```

   where the first symbol is the symbolic representation of a character in the CCS *xxxx*, and the second is the symbolic representation of the character in the UCS-2 CCS.

3. Define the procedure for rebuilding and updating the I18NLIB run-time tables.

   For detailed procedures, see "Updating and Installing the I18NLIB Run-Time Tables" in Section 9.

1.9. Introduction to the I18N Service Library (I18NLIB)

I18NLIB adapts 2200 Series system software and applications programs for use in non-US English operating environments.

- 18NLIB Components
- I18NLIB Tools and Services
- I18NLIB Service Routines

Migrating to 8-Bit Coded Character Sets

I18NLIB is used to migrate files and databases smoothly from 7-bit to 8-bit coded character sets (CCS). These CCSs are processed according to local language and cultural conventions.

Spelling of Non-English Proper Names

CCSs provide I18N applications with the appropriate script for a given natural language or locale. This script includes characters unique to that language or locale.

Sorting of Lists

I18N applications use language and cultural conventions to sort lists correctly. For details, see "Sorting and Collating Within I18N Applications" in Section 8.

1.9.1. Components

These components	Are used to
Exec environment variables	Convey information between an application program and the I18NLIB service routines.
X/Open locale definition files	Define processing rules for a region. Related CHARMAP files define allowable characters for the language of that region.
LOCDEF processor	Transform locale definition files into tables that can be used by the service routines. More about the LOCDEF processor
Service routines	Process data passed by the program according to the rules supplied by the locale definition.
Repository entities and relationships	Used to track and report on locale, CHARMAP, translation, and other information that is available to system software, application programs, and service routines.
COPY procs	Provide declarations for service routines and define templates for structures used by service routines. Procs are available for the C Compiler, the COBOL Compiler, and for PLUS in basic and extended mode.

1.9.2. Tools and Services

The I18NLIB contains	Which are used to
Locale Definition Tools	Define character sets and character conventions.
Browsing Tools	Determine locales and coded character sets (CCS) supported.
Locale Access Methods	Examine and change character sets currently in force.
Data Announcement	Support data identification.
Transliteration Services	Convert data from one CCS to another.
Format Conversion Services	Convert a character to another representation of the character within the same CCS.
Character Classification	Support classification of alpha, numeric, and special characters.
Case Mapping Service	Changing characters from uppercase to lowercase and vice versa.

1.9.3. I18NLIB Service Routines

The I18N Service Library (I18NLIB) is a collection of service routines used to process I18N-related data that is passed by a program according to the rules supplied by a locale definition. These service routines are described by language in Sections 3 through 7.

1.10. Overview of I18N Enhancements

Internationalization (I18N) capabilities have been enhanced over many releases and in many products. This topic provides an enhancement history of I18NLIB by release level.

The following subsections list the enhancements to the I18NLIB product and the product level/system release level in which the capability was introduced. The list is in reverse order (newest information first).

I18NLIB Level 2R2 Enhancements

(ClearPath OS 2200 Release 12.0)

Enhanced STRFTIME Service Routine

The *strftime*() service routine is enhanced to allow the specification of an optional minimum field width specification for the %C, %F, %G, and %Y conversion specification.

For *strftime*(), the format of a conversion specification is:

```
'%' (minimum field width) (modifier) conversion-specifier-character
```

where items in parenthesis are optional. The minimum field width is the new addition and has the following format:

```
[ '0' | '+' ]  min-width
```

The zero character ('0') defines that the padding character will be '0'. The plus sign character ('+') also defines that the padding character will be '0' and that if more than four bytes are required to represent the year (for the %F, %G, and %Y conversion specifications) or two bytes to represent the year divided by 100 (for the %C conversion specification) then a leading plus sign character is included if the year being processed is greater than or equal to zero. The default padding character is undefined.

The *min-width* specification is a series of the digit characters '0' through '9' that defines the minimum field width specification. If the converted value including any leading '+' or '-' sign, has fewer bytes than the minimum field width, the output is padded on the left after any leading '+' or '-' sign) with the '0' padding character.

The minimum field width specification can only be specified for the %C, %F, %G, and %Y conversion specifications without the optional E or O modifier.

In addition to the new minimum field width specification for the year, other numeric conversion specifications have an implicit field width. For the %d, %D, %e, %H, %I, %m, %M, %S, %U, %V, and %W conversion specifications the field width is two bytes. For the %u and %w conversion specifications the field width is one byte. For the %j conversion specification the field width is three bytes.

Enhanced STRPTIME Service Routine

The *strptime*() service routine is enhanced to allow the specification of an optional field width specification for the %C and %Y conversion specifications without the optional E or O modifiers. Also, define all numeric conversion specifications fields as having a maximum number of digits.

For *strptime*(), the format of a conversion specification is:

```
'%' (field-width) (modifier) conversion-specifier-character
```

where items in parenthesis are optional. The *field-width* is the new addition and has the following format:

```
( '0' | '+' )  max-width
```

The zero character, '0', and plus sign character, '+', are optional and are ignored. When the *max-width* field is specified, it is interpreted as a string of decimal digits that determine the maximum number of bytes converted for the conversion rather than the number of bytes defined in the description of the conversion specification. The field-width specification is only allowed for the %C and %Y conversion specifications without an optional E or O modifier.

If the field width specification is not specified, the maximum width for the %C conversion specification is two bytes and four bytes for the %Y conversion specification.

As part of this enhancement, the implicit maximum field width is two for the %d, %e, %H, %I, %m, %M, %S, %U, %W, and %y conversion specifications; one for the %w conversion specification and three for the %j conversion specification.

I18NLIB Level 2R1Enhancements

(ClearPath OS 2200 Release 10.0)

I18NLIB has been enhanced to significantly improve performance for the service routines. The primary emphases for these improvements are the UC isa*xxxx*(), *tolower*(), and *toupper*() routines as well as calls to these service routines from other UCS languages. However, the performance improvements made should improve the performance of the other service routines as well for the UCS language bindings. The non-UCS bindings may also improve slightly. The improvements include the packaging of the I18NLIB service routines in an AFCB versus a subsystem, reduced path length, and reduced IP and Executive service times. The performance improvements will be realized for the non-first call to I18NLIB in an activity once I18NLIB 2R1 is installed. No recompilation, relinking, or recollection is necessary.

As a result of converting the I18NLIB service routines from a subsystem to an AFCB, a migration issue exists if you need to revert back to any I18NLIB 1R3 level after I18NLIB 2R1 has been installed. If you need to revert back to any I18NLIB 1R3 level, then the following steps must be performed:

1. Re-install the I18NLIB 1R3x level with SOLAR / PRODLD.
2. Immediately following the completion of the above install, a system reboot must be performed.

The system reboot is necessary ONLY when reverting back to any I18NLIB 1R3 level after I18NLIB 2R1 has been installed. No reboot is necessary when moving from a I18NLIB 1R3 level to I18NLIB 2R1.

I18NLIB Level 1R3E Enhancements

(ClearPath OS 2200 9.0)

The STRFTIME service routine has been enhanced by adding or updating the %F, %g, %G, %z, and %Z conversion descriptors. The meaning of these conversion descriptors is as follows:

%F Equivalent to %Y-%m-%d (the ISO 8601:2000 standard date format) For example, 2003-07-31 for July 31, 2003.

%g Replaced by the last 2 digits of the week-based year as a decimal number [00-99]. For example, 03 for July 31, 2003.

%G Replaced by the week-based year as a decimal number. For example, 2003 for July 31, 2003.

%z Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm) or by no characters if no timezone is determinable. For example, -0600 for Central Standard Time or -0500 for Central Daylight Time.

%Z Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. For example, CST for Central Standard Time.

I18NLIB Level 1R3C Enhancements

(ClearPath OS 2200 Release 7.1)

Modifications to enable the definition and translation of private use (user-defined) characters to the CCS_TO_CCS service routine. To accomplish this task, the run-time tables need to be updated.

I18NLIB Level 1R3 Enhancements

(HMP IX Release 5.1)

- The addition of a new NAME_AND_NUMBER packet interface service routine that returns the number of bits per character for a requested CCS or CCS associated with a requested locale, as well as other name-and-number information.
- The addition of a new 8-bit character CHARMAP for the Japanese CCS JIS-X0201 and the addition of CHARMAPs for the following 16-bit character sets: UCS-2 (Unicode) and the Japanese CCS LETS-J. This includes addition of a new code type (octal 075) for UCS-2, new LETS-J source code elements for transliteration, and changes to the LOCDEF processor to handle the new CHARMAPs and source code element.
- Enhancements to the OPEN_CCS_TO_CCS and CLOSE_CCS_TO_CCS service routines to enable translation of any installed CCS to UCS-2 and UCS-2 to any installed CCS.
- The conversion of the LOCDEF processor from basic mode to extended mode. Related changes include the following: removal of the date of the executable in the processor call information and production of omnibus elements instead of relocatable elements.

I18NLIB Level 1R2B Enhancements

(HMP IX Release 5.0)

Creation of I18NLIB installation mode B, which defines the default locale as en_US.ASCII and the default code type as ASCII. Installation mode A still exists with the default locale as POSIX and the default code type as ISO 8859-1 (043).

I18NLIB Level 1R2 Enhancements

(HMP IX Release 4.0)

- The addition of four new I18NLIB service routines: STRFMON, STRFTIME, STRPTIME, and NL_LANGINFO.

- The addition of four new categories and their corresponding environment variables: LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.
- Enhancements to three existing I18NLIB service routines: GETENV, GETENVSYS, and SETLOCALE.
- Recognition and processing of the locale definition (LOCDEF) file categories LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.

1.11. Overview of I18N Incompatibilities

Modifications and corrections to the I18NLIB product can cause differences or incompatibilities to all products that have I18N capabilities. This topic provides incompatibility information for products that are described in more detail in Section 8.

Following is a list of incompatibilities in the I18NLIB product and the product level/system release in which the incompatibility was introduced. The list is in reverse order (newest information first). The information for products other than I18NLIB is specific to that product or the products it uses.

I18NLIB Level 2R2 Incompatibilities

(ClearPath OS 2200 Release 12.0)

STRFTIME Service Routine

As a result of the I18NLIB 2R2 enhancement to the *strftime*() service routine, the value for a year can now be less than 0 or greater than 9999. In addition, if a minimum field width is not specified, the minimum length is assumed to be two bytes for the %C conversion specification and four bytes for the %Y conversion specification. These changes might cause some different date / time strings to be created or allow a string to be created when an error would have been returned previously.

STRPTIME Service Routine

As a result of the I18NLIB 2R2 enhancement to the *strptime*() service routine, the value for a year can now be less than 0 or greater than 9999. These changes might cause some date / time strings to be parsed differently or allow a string to be parsed when previously an error would have been returned. Also, due to the implicit maximum field width being defined for many conversion specifications, different results or an error could be returned. The different results would occur only if the actual field width is greater than the defined implicit maximum field width.

I18NLIB Level 2R1 Incompatibilities

(ClearPath OS 2200 Release 10.0)

The name and location of the element used to resolve the I18NLIB entry points for OMs has changed. If any static link directive or SSDEF directive contained an explicit reference to the file SYSLIB*I18NLIB-GDEF or the element SYSLIB*I18NLIB-GDEF.GATEDEF it will fail. The preferred method to reference I18NLIB in these circumstances is via the libcodename I18NLIB which has not changed.

For the UC binding to the getenv(), nl_langinfo(), and setlocale() service routines, the string returned by a successful call will be overwritten on a subsequent call to the same service routine. This is consistent with the definition of these C functions. In I18NLIB 1R3x, the string value returned was not overwritten on a subsequent call to the same service routine. If it is necessary to save the string value returned over multiple function calls, a strcpy() function must be used to copy the string to a new location.

The format of the runtime tables has changed from I18NLIB 1R3x to I18NLIB 2R1. As released, the runtime table contains all of the locales and coded character set (CCS) transliteration tables supported by I18NLIB. If the I18NLIB 1R3 runtime tables were modified from the released version, then these modifications must be performed again through the runtime table build process. A saved copy of the I18NLIB 1R3x runtime tables will not work with I18NLIB 2R1 and will produce an error status or cause a contingency to be returned. Likewise, the I18NLIB 2R1 runtime tables are incompatible with I18NLIB 1R3x. Whenever an install of I18NLIB is performed, the correct complete version of the I18NLIB runtime tables is always installed.

The error status returned by any I18NLIB 2R1 service routine may be different from the error status returned by the I18NLIB 1R3x service routine if multiple error conditions exist on a particular call. This should not have an impact unless you were performing a function and expected an explicit error status to be returned and multiple errors actually existed.

In I18NLIB 2R1, several small defects were corrected. Some results returned by I18NLIB 2R1 may be different from I18NLIB 1R3F. It is also possible that a service routine may return an error when I18NLIB 1R3F did not detect an error. Likewise, the reverse is also true; I18NLIB 1R3F may have detected an error when in fact no error existed and I18NLIB 2R1 now does not return an error status.

When a status other than OK is returned, the contents of output buffers, strings, or variables may be different from I18NLIB 1R3F if no explicit definition of these variables is defined for an error case. In all such cases, the content of these items is undefined.

I18NLIB 2R1 requires URTS 13R1B to be installed on Exec 47R4 at a minimum. Also, if you use FLIT to execute a program that calls I18NLIB directly from the program or indirectly by calling a product that calls I18NLIB, FLIT 13R1 may be required.

As a result of converting the I18NLIB service routines from a subsystem to an AFCB, a migration issue exists if you need to revert back to any I18NLIB 1R3 level after I18NLIB 2R1 has been installed. If you need to revert back to any I18NLIB 1R3 level, then the following steps must be performed:

1. Re-install the I18NLIB 1R3x level with SOLAR / PRODLD.
2. Immediately following the completion of the above install, a system reboot must be performed.

The system reboot is necessary ONLY when reverting back to any I18NLIB 1R3 level after I18NLIB 2R1 has been installed. No reboot is necessary when moving from a I18NLIB 1R3 level to I18NLIB 2R1.

I18NLIB Level 1R3F Incompatibilities

(ClearPath OS 2200 Release 9.1)

No incompatibilities were introduced in I18NLIB 1R3F.

I18NLIB Level 1R3E Incompatibilities

(ClearPath OS 2200 9.0)

The STRPTIME service routine will now correctly parse a string time format into the appropriate time structure variables. Prior levels had several minor defects and, as a result, different results may be received. A specific change to the STRPTIME %y conversion descriptor was also made so that the two-digit year range will now represent 1968-2067 instead of 1900-1999. Please note that the changes to STRPTIME may also be seen in the STRFTIME service routine, if the two service routines are used together.

I18NLIB Level 1R3D Incompatibilities

(ClearPath OS 2200 8.0)

The run-time tables for levels 1R3, 1R3A, 1R3B, and 1R3C for the TOUPPER service routine were incorrect if a one-to-two character mapping existed. The TOUPPER service routine will now properly convert those one-to-two character conversions. Correcting the conversion might cause a compatibility issue if the prior incorrect conversions were used and stored in a database or file. It is also possible that the call to the TOUPPER service routine might now return an error status that was not applicable before (for example, levels 1R3-1R3C did not convert the character or converted it to the first character of the two uppercase characters).

The locales affected by this problem are

- de_AT.8859-1
- de_CH.8859-1
- de_DE.8859-1

The CCS_TO_CCS, CONVERT_FORM, TOLOWER, and TOUPPER service routines will return fatal error status 17 if any part of the TO character string buffer overlaps any part of the FROM character string buffer. Previous levels of I18NLIB did not detect this condition.

UCS based transactions that call I18NLIB services routines will no longer lose sticking power based on the use of I18NLIB.

I18NLIB Level 1R3C Incompatibilities

(ClearPath OS 2200 Release 7.1)

The CCS_TO_CCS service routine incorrectly returned the number of characters instead of the number of bytes in the TO_LENGTH parameter when the TO_CCS was a 2-byte

CCS. This incompatibility applies only to the COBOL Compiler and the basic mode and extended mode PLUS bindings for the CCS_TO_CCS service routine.

I18NLIB Level 1R3B Incompatibilities

(ClearPath OS 2200 Release 7.0)

The collation order of the following locales have changed as a result of a change in the en_DK.8859.1 (English used in Denmark) locale definition file:

Locale Name	Language/Where Used	Associated Charmap	Locale Number
de_AT.8859-1	German/Austria	ISO8859-1	3
de_CH.8859-1	German/Switzerland	ISO8859-1	4
de_DE.8859-1	German/Germany	ISO8859-1	5
de_LU.8859-1	German/Luxembourg	ISO8859-1	6
el_GR.8859-7	Greek/Greece	ISO8859-7	7
en_DK.8859-1	English/Denmark	ISO8859-1	9
en_GB.8859-1	English/Great Britain	ISO8859-1	10
en_IE.8859-1	English/Ireland	ISO8859-1	11
fr_BE.8859-1	French/Belgium	ISO8859-1	17
fr_CH.8859-1	French/Switzerland	ISO8859-1	19
fr_FR.8859-1	French/France	ISO8859-1	20
fr_LU.8859-1	French/Luxembourg	ISO8859-1	21
it_IT.8859-1	Italian/Italy	ISO8859-1	25

The following are affected by these collation ordering changes and must be updated:

- The collation order string produced by the STRING_TRANSFORM service routine. For instructions on updating ordering keys and what happens to previously sorted data, see "STRING TRANSFORM: Updating an Ordering Key."
- Any I18N column in the Relational Database Server (formerly RDMS 2200) that uses one of the changed locales. For instructions on what to do with tables containing these columns see "Relational Database Server, Impact of Locale Collation Difference on I18N Columns."

I18NLIB Level 1R3A Incompatibilities

(HMP IX Release 6.1)

No incompatibilities were introduced in I18NLIB 1R3A.

I18NLIB Level 1R3 Incompatibilities

(HMP IX Release 5.1)

In previous versions of I18NLIB, some characters were incorrectly defined, translated to incorrect values, or not defined in the Unisys internal transliteration character set. These problems have been corrected in level 1R3 and as a result, some transliterations done by way of the CCS_TO_CCS service routine bindings may produce different results.

The following cases will have different transliterations based upon the CCS. The characters that are affected are identified by their name and the encoding value. The encoding value is either the octal CCS encoding or the 16-bit hexadecimal Unicode encoding (specified as Uxxxx).

BICS CCS (code type 057)

Many characters that were not defined for this CCS have now been defined. As a result

- When the BICS CCS is the FROM CCS, the previously undefined characters were always translated into the TO CCS replacement character. If the character is defined in the TO CCS, the character will now translate to that character.
- When the BICS CCS was the TO CCS, any of these characters would have translated into the BICS replacement character, LOW LINE (0137). The character now translates into the appropriately defined character.

EBCDIC-US CCS

Many characters that were not defined for this CCS have now been defined. As a result

- When the EBCDIC-US CCS is the FROM CCS, the previously undefined characters were always translated into the TO CCS replacement character. If the character is defined in the TO CCS, the character will now be properly translated to that character.
- When the EBCDIC-US CCS was the TO CCS, any of the characters previously undefined would have translated into the EBCDIC-US replacement character, LOW LINE (0155). The character now translates into the appropriately defined character.
- In addition, the characters NOT SIGN (0260), QUOTATION MARK (0177), LEFT SQUARE BRACKET (0255), RIGHT SQUARE BRACKET (0275), and CIRCUMFLEX (0137) were all improperly defined. When EBCDIC-US was the FROM CCS, these characters were incorrectly translated to other characters. When EBCDIC-US was the TO CCS, these characters were translated to the wrong character. The character now translates into the appropriately defined character.

ISO646CA CCS (code type 032)

- When this CCS is the FROM CCS, the character QUESTION MARK (077) may be incorrectly converted in the TO CCS.
- When this CCS is the TO CCS and the character being translated is a COMMERCIAL-AT (U0040), LEFT SQUARE BRACKET (U005B), REVERSE SOLIDUS (U005C), RIGHT SQUARE BRACKET (U005D), CIRCUMFLEX ACCENT (U005E), GRAVE ACCENT (U0060), LEFT CURLY BRACKET (U007B), VERTICAL LINE (U007C), RIGHT CURLY BRACKET (U007D), or TILDE (U007E), the character was

converted to a NULL (0) character instead of the replacement character LOW LINE (0137).

ISO646ES CCS (code type 027)

When this CCS is the TO CCS and the character being translated is a NUMBER SIGN (U0023), COMMERCIAL-AT (U0040), LEFT SQUARE BRACKET (U005B), REVERSE SOLIDUS (U005C), RIGHT SQUARE BRACKET (U005D), LEFT CURLY BRACKET (U007B), VERTICAL LINE (U007C), or RIGHT CURLY BRACKET (U007D) the character was converted to a NULL (0) character instead of the replacement character LOW LINE (0137).

ISO646IT CCS (code type 026)

When this CCS is the TO CCS and the character being translated is a NUMBER SIGN (U0023), COMMERCIAL-AT (U0040), LEFT SQUARE BRACKET (U005B), REVERSE SOLIDUS (U005C), RIGHT SQUARE BRACKET (U005D), GRAVE ACCENT (U0060), LEFT CURLY BRACKET (U007B), VERTICAL LINE (U007C), RIGHT CURLY BRACKET (U007D), or TILDE (U007E), the character was converted to a NULL (0) character instead of the replacement character LOW LINE (0137).

ISO646PT CCS (code type 034)

When this CCS is the TO CCS and the character being translated is a COMMERCIAL-AT (U0040), LEFT SQUARE BRACKET (U005B), REVERSE SOLIDUS (U005C), RIGHT SQUARE BRACKET (U005D), LEFT CURLY BRACKET (U007B), VERTICAL LINE (U007C), RIGHT CURLY BRACKET (U007D), or TILDE (U007E), the character was converted to a NULL (0) character instead of the replacement character LOW LINE (0137).

ISO8859-10 CCS (code type 054)

When this CCS is the FROM or TO CCS, the character EM DASH (0274) is now properly defined and will convert to the proper value. In earlier releases, the EM DASH was converted to the LOW LINE (U005F) character.

ISO8859-5 CCS (code type 047)

When this CCS is the FROM or TO CCS, the character NUMERO SIGN (0360) is now properly defined and will convert to the proper value. In earlier releases, the NUMERO SIGN was converted to the LOW LINE (U005F) character.

ISO8859-7 CCS (code type 051)

When this CCS is the FROM or TO CCS, the characters SINGLE HIGH-REVERSED-9 QUOTATION MARK (0241), RIGHT SINGLE QUOTATION MARK (0242), and EM DASH (0260) are now properly defined and will convert to the proper value. In earlier releases, these characters were converted to the LOW LINE (U005F) character.

ISO8859-8 CCS (code type 052)

When this CCS is the FROM or TO CCS, the character DOUBLE LOW LINE (0337) is now properly defined and will convert to the proper value. In earlier releases, the DOUBLE LOW LINE was converted to the LOW LINE (U005F) character.

ISO8859-1 and Collating Character Coding

Many ISO8859-1 based locales will collate character encoding 0257 differently than in previous levels. The ISO8859-1 CHARMAP incorrectly identified character encoding 0257 as the OVERLINE character (U203E) instead of the MACRON character (U00AF). Thus, character encoding 0257 will now collate before many other characters. As a result of correcting character encoding 0257 different results could be encountered.

The following are affected by these collation ordering changes and must be updated:

- The collation order string produced by the STRING_TRANSFORM service routine. For instructions on updating ordering keys and what happens to previously sorted data, see "Updating an Ordering Key" in the STRING_TRANSFORM Service Routine topic in Section 5.
- Any I18N column in the Relational Database Server (formerly RDMS 2200) that uses one of the changed locales. For instructions on what to do with tables containing these columns see "Impact of Locale Collation Difference on I18N Columns" in the Relational Database Server topic in Section 8.

I18NLIB Level 1R2B Incompatibilities

(HMP IX Release 5.0)

No incompatibilities were introduced in I18NLIB Level 1R2B.

I18NLIB Level 1R2A Incompatibilities

(HMP IX Release 4.1)

No incompatibilities were introduced in I18NLIB Level 1R2A.

I18NLIB Level 1R2 Incompatibilities

(HMP IX Release 4.0)

No incompatibilities were introduced in I18NLIB Level 1R2.

Section 2
Preparing for Internationalization

2.1. Making the Transition to the I18N Environment

Unisys OS 2200 systems are already multi-CCS systems because they contain data encoded in both Fieldata and ASCII.

With I18N features added, multiple CCS-support has been extended to include service routines and other capabilities that allow you to more easily create applications that use CCSs other than Fieldata and ASCII.

2.1.1. Transition Paths

If you select this path	You complete the transition tasks in this order	Which has these advantages
Evolutionary	In stages until all system components support the selected CCS. Transliterate and reload your files and data bases. See 2.1.3.	This method works best when • Multiple suites of applications must be adapted to I18N processing. • Critical applications cannot be unavailable for long periods of time. • Administrative oversight is available for a long transition period.
Migration to I18N Coded Character Sets (CCS)	First, perform transition tasks 1, 2, 3, 5, and 6 (see 2.1.2). Then use the CONVERT_FORM service routine to aid in task 4.	This method works best when • Sites have familiar, well-understood applications. • Critical applications cannot be unavailable long enough to perform a Big Bang transition. • Administrative oversight is available for a long transition period.
Big Bang	All at the same time. Transliterate and reload your files and data bases.	This method works best when • Sites have just a few familiar, well-understood applications. • Critical applications can be unavailable long enough to make the transition. • Administrative oversight of the transition must be minimized.

2.1.2. General Transition Tasks

Task 1: Determine which CCS best fits your data storage and processing needs.

Task 2: Determine the corresponding locale that best fits your cultural convention requirements.

Task 3: Replace 7-bit terminals and printers with 8-bit terminals and printers.

Task 4: Move your file and database data from 7-bit to 8-bit CCSs. You can transliterate the data and reload it, or you can use the CONVERT_FORM service routine.

Task 5: Update applications to handle the new terminals, printers, and CCSs.

Task 6: Add the I18NLIB service routines to support culturally-sensitive processing.

2.1.3. Sample Evolutionary Transition

Note: *The following example is fictitious; no resemblance to any Unisys client is intended.*

Assume the following parameters for an existing system:

- An insurance company in Germany serves 5000 clients in Germany and France. It has two offices, one in Germany and one in France.
- Only two kinds of policies are issued: fire insurance for homes and accident insurance for automobiles.
- The company's processing applications use the COBOL Compiler, the Display Processing System, and the Network Database Server.
- The company does not use a National Replacement Character Set (NRC).
- The database for the Network Database Server keeps track of information in three sets.
  - Client information: Name, address, telephone number, pointers to home and auto policies
  - Home policy: Policy number, property address, coverage amount
  - Auto policy: Policy number, vehicle identification number, coverage amount
- The company uses two distributed communications processors (DCP), one in each office, and 200 UTS terminals, 100 in each office.

2.1.4. Step-by-Step Transition Procedure

Steps for Insurance Company Example

Follow these steps to complete data transition, terminal replacement, and application updates in stages until all system components support the I18N features.

1. Determine which natural language or languages have to be supported.

   Since the company's clients are in Germany and France, the language and cultural conventions of the German and French languages need to be supported.

2. Select the coded character set (CCS) that supports the language or languages.

   Both German and French are supported by the Latin-1 character set in the ISO 8859 suite of character sets, so the ISO 8859.1 CCS should be selected.

3. Install hardware that supports 8-bit character sets.

   The UTS terminals support only 7-bit character sets, so they will need to be replaced by terminals that support 8-bit character sets, such as the Unisys LT300 terminal.

   Supporting printers will also be needed.

4. Update software to run on the new terminals.

   The file and database data need to be moved from 7-bit to 8-bit CCSs. You can transliterate the data and reload it, or you can use the CONVERT_FORM service routine.

   Each application will need to be updated to handle the new terminals, printers, databases, and CCSs.

2.2. Migrating Hardware to the I18N Environment

2.2.1. Unisys Terminal Considerations

Note: *The Display Processing System was formerly known as DPS 2200, which remains the installation and support name.*

Add 8-Bit Terminals or Terminal Emulators

Site administrators need to add 8-bit terminals or terminal emulators to allow for the input, display, and transmission of 8-bit characters.

The Unisys LT300 terminal has been internationalized to handle 8-bit characters, which includes support for many of the ISO 8859 character sets.

However, existing devices such as the UTS 20 and UTS 400 terminal classes can handle only the 7-bit ISO 646 national replacement characters (NRC).

Enhance Applications to Handle 8-Bit Terminals and Display Processing System Messages

An OS 2200 application must be enhanced to identify and handle 8-bit terminals so that it can build messages and include control sequences for this terminal class.

The LT300 terminal can use the Display Processing System to manage screen-handling transparent to the application, but the application must be able to generate Display Processing System message text in the appropriate 8-bit coded character set (CCS).

2.2.2. Printer Considerations

Note: *The Enterprise Output Manager was formerly known as DEPCON, which remains the installation and support name.*

Know Your Printer Capabilities

The format of printer output is controlled by the application, not the system. Know which coded character sets (CCS) your printers can process, as well as the appropriate printer control language.

Know Enterprise Output Manager Features

The Enterprise Output Manager application recognizes CCS-IDs in print files. Enterprise Output Manager features include

- Support for the translation tables that map between file CCSs and printer CCSs.
- Use of CCS tables to determine what actions to take for a given CCS.

2.2.3. Disk Storage Implications

No Extra Disk Space Required

The OS 2200 architecture treats both 7-bit and 8-bit fields as a quarter-word, that is, 8-bit data does not require any more disk space than 7-bit data.

Modify Data Encoded in ASCII

If the data is encoded in ASCII, declare it to be a specific ISO 8859 character set.

Transliterate Diacritical Characters

If the data is encoded using an ISO 646 national replacement character set (NRC), transliterate any characters with diacritical marks to ISO 8859.*n* encodings.

2.3. Migrating Software to the I18N Environment

I18N capabilities support end-to-end 8-bit processing and delivery of 8-bit characters to user applications from the terminal through the communications, screen-handling, and operating system software.

Verify the Transition

After internationalizing your user applications, check them to verify the effects of the standard 8-bit character sets and the new SDF code types.

An I18N application must be able to handle the 8-bit characters required by the application without character corruption. The eighth bit cannot be ignored or used for some other control purpose.

Where appropriate, an I18N application must able to handle values other than 0 or 1 in the code-type field of the SDF image control word.

As part of the verification process, you may then need to recompile, recollect, and relink these applications.

Modify Testing Requirements

Verifying I18N applications is not a one-time operation. Any tests that you create or modify need to be added to your suite of standard regression tests.

The testing requirements depend on the type of processing performed by an application.

Valid Locales for Migrating

The following locales have been developed for migration and are currently supported by the I18NLIB product. For more details, go to the "Locale Names and Numbers Used With the CONVERT_FORM Service Routine" topic in Appendix B.

Locale Name	Language/Country	Associated Charmap	Locale Number
da_DK.MIG646DK*	Danish/Denmark	MIG646DK	44
de_DE.MIG646DE*	German/Germany	MIG646DE	47
en_CA.MIG646CA*	English/Canada	MIG646CA	49
es_ES.MIG646ES*	Spanish/Spain	MIG646ES	52
fr_CH.MIG646FR*	French/Switzerland	MIG646FR	56

Crossboot Restrictions

Files and databases can become corrupted if, after I18N applications are installed, the Exec or certain other products are crossbooted to a level that does not support I18N capabilities.

Network Database Server Schema

When an application accesses an internationalized Network Database Server schema, the schema will abort at IMPART time.

Relational Database Server Collation

If a Relational Database Server collation does not have a valid name value or an associated character set, the collation cannot take place.

SDF Files

The SDF files recovered on the system may contain data or control images that are tagged with 8-bit CCS types in a 7-bit environment.

Migrating Specific Products

Migrating INFOConnect (see 2.3.1)

Migrating IS-6000 (see 2.3.2)

Migrating LINC II (see 2.3.4)

Migrating Business Information Server (MAPPER) (see 2.3.3)

Migrating Hardware (see 2.2)

2.3.1. Migrating INFOConnect to an I18N Environment

Determine the CCS

The INFOConnect character set is determined by the version of Windows that you are using (currently available in Arabic, Greek, and eastern European languages as well as for English). An ISO 8859 coded character set (CCS) is always used.

Localize the Files

INFOConnect has its own localization kit that includes all the necessary files and instructions. To obtain this kit, contact your Unisys representative.

2.3.2. Migrating IS-6000 to an I18N Environment

If you already use the IS-6000 terminal emulator, you do not need any more expertise to install the I18N version, but there are additional installation and configuration tasks.

Manually Configure Terminal CCSs

The I18N version will not impact your workflow, but you may need to configure a terminal CCS manually in some environments.

Install and Configure Communications Software

To use the I18N version, you need to install and configure compatible levels of DCP Series Telcon, Telcon program products, and CMS 1100.

2.3.3. Migrating the Business Information Server to an I18N Environment

Note: *The Business Information Server was formerly known as MAPPER, which remains the installation and support name.*

Modify Data-Tagging Capabilities of Business Information Server Runs

The Business Information Server does not tag data internally. If you want to use this server in the I18N environment, modify the runs to tag data at the application level.

Keep the CCS Identifier in the Report Header

This allows an I18N Business Information Server application to

- Detect the CCS ID of data entering the server system.
- Specify the appropriate CCS ID in the character set field.

Ensure Terminal Support of Specified Coded Character Sets

An I18N application on the Business Information Server uses the reserved words TCCS$ for terminal input and STAT3$ for file input. If your terminal does not support the CCS of the application data, the data will be corrupted.

2.3.4. Migrating LINC II to an I18N Environment

Recode Applications to Support Multiple CCSs

If you want to use the LINC II system in the I18N environment, you may need to recode some of the existing applications if they are required to support multiple CCSs.

Since the database software on the Network and Relational Database Servers supports multiple collating sequences based on locales, you do not need to provide logic for handling special cases of nonbinary collating of data stored in LINC II databases.

List of LINC II Migration Tasks

Because an I18N LINC II application can	You need to	To ensure that
(1) Provide the coded character set identifier (CCS ID) of data coming from an I18N terminal in a LINC II system setup data item.	(1a) Decide whether to store the 8-bit data in its current form or translate it. (1b) Provide logic to determine if a given record should be displayed on a specific terminal.	(1) A given record is displayed, based on the terminal type and the CCS it is transmitting.
(2) Handle storing and retrieving of multiple CCSs in the same database.	(2) Decide if you want to store multiple CCSs in the same database. If so, add a field to the database record where the CCS ID can be stored (LINC item GLB.CCS).	(2) The system knows the CCS when retrieving data from this database.

2.4. Identifying the CCS of a Character When Reading Images

Source of Data

- From a TERMINAL
- From a SYSTEM CONSOLE
- From a FILE
- From a DATABASE
- From a NETWORK

2.4.1. Identifying the CCS for an Image Read from a Terminal

If you are operating in	Identify the CCS by using
Basic Mode	ER SYMB$ ER RSI$ SYSLIB SAR$ READ SYSLIB SIR$

If you are operating in	Identify the CCS by using
Extended Mode	Compiler language statements SLIB S$SREAD
Basic or Extended Mode	MCB, TIP/HVTIP, and COMPOOL functions Display Processing System

ER SYMB$

Changes to ER SYMB$ Read Function

The ER SYMB$ read function includes 8-bit character set support for the ISO 646 and ISO 8859.n coded character sets (CCS).

If the UNTR$ and SPEC$ mode bits are	The following results occur:
Set	The redefined character set values are returned
Not set	The result depends on the character type of the image being read and on whether the image needs to be converted to Fieldata. • The caller is alerted when the data being returned has originated from a CCS other than FIELDATA, ASCII/ISO, or ASCII/APL. • If the first attempt to read Fieldata or ASCII-like images fails, an 02 warning status is returned in S1 of Word 3 of the SYMB$ packet. • If subsequent attempts of the same type fail, the result is an error type that identifies the CCS of the incoming data.

Location of CCS Identifier for Incoming Image

If ER SYMB$ is used, the CCS identifier for the incoming image is returned in the character set field of the SYMB$ packet. The values for this field match the character set types for SDF files.

Use of ER RSI$ for Image Identification

ER SYMB$ functions rely on the image identification capabilities of ER RSI$.

ER RSI$

Returning Terminal Type

ER RSI$ returns the terminal type in the character set field of the RSI$ packet.

Handling Images From Runstream or @ADD Files

If the images come from a runstream file or an @ADD file, RSI$ can pass the CCS identifier for the SDF image to the application.

SYSLIB SAR$ READ

Determining the CCS Identifier and Controlling Return of Images

The CCS identifier of each image is returned in the read packet. You can use additional settings in this packet to control the return of images in various CCSs.

Settings for Read Packet Fields

If the calling program is	The CCS identifier is returned in this field	As these image types	If you set this field	To
Basic Mode PLUS	CHARACTER_TYPE (CHARACTER_CODE)	All CCSs	UNTRANSLATE_MODE	ON
		Fieldata, ASCII-like, or ACD only		OFF
Basic Mode MASM	SR$CHARTYP	All CCSs	SR$UNTRFLG	1
		Fieldata, ASCII-like, or ACD only		0

Using the SDF Control Word Field (Optional)

When reading SDF files and elements, you do not need to examine the SDF control words to determine the CCS of input images. Nevertheless, you may still request that SDF control records be returned so that you can view them in the read packet.

Process for Returning SDF Control Records

Process Steps	Basic Mode PLUS	Basic Mode MASM
To specify which SDF control records to pass, use	SDF_CONTROL_RECORDS_TO_PASS	SR$SDFCNTRL
• The entire SDF control word is returned in	SDF_IMAGE_CONTROL_WORD	SR$SDFICW
• The type of the SDF control word is returned in	CONTROL_RECORD_TYPE	SR$ICWTYPE
• The CCS identifier for certain control records is returned in	CONTROL_RECORD_CHAR_TYPE	SR$ICWCHAR
• The CCS identifier for certain data records is returned in	DATA_RECORD_CHAR_TYPE	SR$ICWCHAR

Process Steps	Basic Mode PLUS	Basic Mode MASM
• The CCS identifier of the most current Label Control Record or Character Set Change Control Record is returned in	INPUT_CHARACTER_CODE	SR$INCHARTYP

SYSLIB SIR$

Requirements for Returning a CCS

A combination of SIR$ initialization options and specific SIR$ requests is required to correctly identify the CCS of each image returned from a terminal or SDF file/element.

Process for Returning a CCS

1. Initialize SIR$.
   a. Use the INISR$ entry point to initialize SIR$.
   b. Set one of the following bits of the initialization word if you want the CCS of the input images to be returned. (The initialization word is provided in register A4 during a call to the SIR$ initialization entry point INISR$.):
      – For Fieldata and ASCII-like images, set Bit 1. Bit 1 is set to cause the CCSs to be returned along with the images. If Bit 1 is not set, the CCS is not returned with the image.
      – For images in all CCSs, set Bit 2. Bit 2 is set to cause images in all CCSs to be returned along with the CCS number. If Bit 2 is not set, only Fieldata and ASCII-like images are returned.
2. Obtain input images and CCS numbers.

   Use **one** of the following entry points to obtain input images and their CCS numbers.
   – If you use the entry point GETNM$ and the conditions in Step 1 are met, the CCS of the input image is returned in S6 of register A0.
   – If you use the entry point GETAS$, this data is returned only if the images are Fieldata or ASCII-like.

Processing SDF File/Element (Source Input) Images

The source input images of SDF files and elements processed by SIR$ may be in any CCS. However, to return these images in all CCSs, the required process must be used (see previous paragraph).

Processing Terminal (Runstream) Correction Images

Runstream correction images are merged with source input images to product source output images. Certain CCS-related restrictions may apply.

Formats for Runstream Correction Images

If the format is	The following function is performed:	And these CCS-related restrictions apply
-n, m	Delete images	The correction image must be Fieldata or ASCII-like.
-n, m *new-image* . . .	Replace images	The correction image must be Fieldata or ASCII-like. New images may be in any CCS.
-n *new-image* . . .	Insert images	The correction image must be Fieldata or ASCII-like. New images may be in any CCS.
-n- *editing-statements* . . .	Edit a single image	The correction image, editing statements, and source input line number n must be Fieldata or ASCII-like images.
-n,m- *editing-statements* . . .	Edit multiple images	The correction image, editing statements, and source input line numbers n through m must be Fieldata or ASCII-like images.

Compiler Language Statements

Direct Access Not Available

Compiler language statements cannot identify the CCS directly for data received from a terminal. Use one of these mechanisms to access the underlying capabilities.

If operating in this mode	Use one of these mechanisms
Demand	SLIB S$SREAD Display Processing System
TIP/HVTIP	MCB, TIP/HVTIP, and COMPOOL functions Display Processing System

SLIB S$SREAD

Determining the CCS Identifier and Controlling Return of Images

The CCS identifier of each image is returned in the read packet. You can use additional settings in this packet to control the return of images in various CCSs.

Settings for Read Packet Fields

The CCS identifier is returned in this field	As these image types	If you set this field	To
Character_Type	All CCSs	Translate_Mode	Do_Not_Translate
	Fieldata, ASCII-like, ACD, and kanji (Japanese) only		Translate

Using the SDF Control Word Field (Optional)

When reading SDF files and elements, you do not need to examine the SDF control words to determine the CCS of input images. Nevertheless, you may still request that SDF control records be returned so that you can view them in the read packet.

Process for Returning SDF Control Records

Process Steps	Read Packet Fields
To request SDF control records to be returned, use	SDF_Control_Records_To_Pass
- When SDF control records are returned, the record type is contained in	Control_Record_Type
- The CCS identifier for certain SDF control records is returned in	Control_Record_Character_Type

MCB

Returning CCS Identifier for TIP/HVTIP Applications

MCB uses the P$CCS field in the CMS Auxiliary Area to return the CCS identifier for an incoming message from a terminal. The values for this field match the character set types for SDF files.

For a list of the currently defined values, refer to the *Message Control Bank* (*MCB) Programming Reference Manual* (7833 1568).

Transliterating Incoming Images for Applications Using COMPOOL

The MCB transliteration service provides drivers to transliterate images from ISO 8859 terminals to an ISO 646 national variant.

These drivers also help to migrate an existing terminal network with its related application programs toward supporting an ISO 8859 character set.

2.4.2. Identifying the CCS for an Image Read from a System Console

Limited Use of ER COM$ and SYSLIB SAR$ COM

These interfaces can be used to return an image from a system console. Both allow a basic mode caller to specify either Fieldata or ASCII for the image to be returned. However, there is no way to identify the CCS of the original image.

2.4.3. Identifying the CCS for an Image Read from a File

If you are operating in	Identify the CCS by using
Basic Mode	SYSLIB SDFI SYSLIB SAR$ READ SYSLIB SIR$
Extended Mode	Compiler language statements SLIB S$SREAD SLIB S$SDF$I

2.4.3.1. SYSLIB/SLIB: Read/Write Functions for SDF Files/Elements

SYSLIB Routines (Basic Mode)

For READ functions	For WRITE functions
SYSLIB SDFI	SYSLIB SDFO
SYSLIB SAR$ READ	SYSLIB SAR$ WRITE
SYSLIB SIR$	

SLIB Services (Extended Mode)

For READ functions	For WRITE functions
SLIB S$SREAD	SLIB S$SWRITE
SLIB SSDFI	

2.4.3.2. SYSLIB SDFI and SLIB SSDFI: Identifying the CCS for an Image Read from an SDF File/Element

Using the SDF File Control Word Field

If you are operating in	The SDF control word for each image (record) read is returned by	In the following location:
Basic Mode	SYSLIB SDFI	File Control table (FCT)
Extended Mode	SLIB SSDFI	Control Word parameter

Certain SDF control words for control records and data records contain the CCS identifier of either the current image or the images that follow.

CCS Identifiers in SDF Control Words

Each image in an SDF file or element contains a control word that indicates whether the image is a control record or a data record.

- A control record contains special control information.
- A data record contains a variable length image.

SDF Control Records: List of Control Record Types

- Location of Control Record Type

  The control record type is contained in S1 (bits 0-5) of the SDF control word.

- Location of CCS Identifier

  The following control records contain the CCS identifier in S6 (bits 30-35) of the SDF control word for the following types of control records (images).

Control Record Number (Octal)	Control Record Name	Description
42	Character Set Change	CCS identifier of the data images that follow
50	Label	CCS identifier initially associated with the SDF file/element
51	Continuation	CCS identifier of this image for this symbiont file
60	Print Control	CCS identifier of the print control image for this symbiont file
61	Special Print Control	CCS identifier of the print control image for this symbiont file
70	Punch Control	CCS identifier of the punch control image for this symbiont file

SDF Data Records: List of SDF File Types

- Location of CCS Identifiers

  Certain SDF file types contain the CCS identifier for each date record (image) in S6 (bits 30-35) of the SDF control word. The SDF file type can be determined from the label control record, the first record in every SDF file.

- Location of Label Control Record

  The label control record (octal 50) is contained in S1 (bits 0-5) of the SDF control word for the first record (image) in the file.

- Location of Fieldata Identifier

  The following data records contain the Fieldata identifier in bits 12-17 of the label control record.

File Type	Fieldata Identifier
READ$ symbiont files	C
@FILE files	I
PRINT$ symbiont files	P
PUNCH$ symbiont files	C

Compiler Language Statements: Identifying the CCS for an Image Read from a File

- COBOL Compiler Support

  The COBOL compiler supports retrieving the CCS identifier for a file and enables an I18N application to retrieve the locale identifier for a file.

- Using SLIB for Direct Access

  The C Compiler and Extended Mode PLUS language statements cannot identify the CCS directly, so the application must use SLIB S$SREAD or SLIB S$SDF$I to access the underlying capabilities.

2.4.4. Identifying the CCS for an Image Read from a Database

If you are operating in	Identify the CCS by using
Basic or Extended Mode	Network Database Server Relational Database Server PCIOS SFS 2200

Network Database Server: Identifying the CCS for an Image Read from a Network Database Server Database

Use of Schema Definition Source

There is no programmatic way of determining the CCS for an image (data record) in a Network Database Server database. The application programmer must consult the schema definition source.

Use of SETLOCALE Service Routine

When the data management routine (DMR) is invoked, it uses the SETLOCALE service routine to establish the current processing environment. It leaves the locale set when it returns control to the application.

The DMR calls SETLOCALE only when it performs culturally-sensitive processing, such as a compare operation during DMR command processing.

Relational Database Server: Identifying the CCS for an Image Read from or Written to a Relational Database Server Database

Determining the CCS Identifier

If the column is a CHARACTER with CHARACTER SET column, the application programmer can consult the table definition to determine the CCS defined for the column.

PCIOS: Identifying the CCS for an Image Read from a PCIOS File

Location of CCS Identifier

The CCS identifier is in S6, the ASCII flag field (also known as the character set field) of the control word for the file header image (SDF 050 image). The values for the CCS identifier match the character set types for SDF files.

Setting the FCFD Field for SSDF and DSDF Files

When an application opens a file for input, PCIOS sets the FCFD field in the File Control Block of the File Control Table based on the ASCII flag field in the file header image (SDF 050 image).

ASCII Flag	FCFD value
0	1 (Fieldata)
1	0 (ASCII/ISO)
2	2 (ASCII/APL)
3 – 77	3 – 77

Setting the FC$LOCALE Field for MSAM Files

When an application opens a file for input, PCIOS sets the FC$LOCALE field in the File Control Block to the locale identifier from the LABLOCALE field of the MSAM Label Area.

From the locale identifier, the application can determine the CCS of the an image by calling the LOCALE_NUM_TO_CCS_NUM service routine.

SFS 2200: Identifying the CCS for an Image Read from an SFS File

Location of CCS Identifier

The CCS identifier is in S6, the ASCII flag field (also known as the character set field) of the control word for the file header image (SDF 050 image). The values for the CCS identifier match the character set types for SDF files.

Setting the FCFD Field for Previously Released Interfaces

When an application opens a file for input and a previously released interface is being used, SFS 2200 sets the FCFD field in the File Control Block of the File Control Table based on the ASCII flag field in the file header image (SDF 050 image) as follows:

ASCII Flag	FCFD value
0	1 (Fieldata)
1	0 (ASCII/ISO)
2	2 (ASCII/APL)
3 – 77	3 – 77

Setting the ST$LOCALE Field for MSAM Files

When an application opens a file for input, SFS 2200 sets the ST$LOCALE field in the Status Message Area to the locale identifier from the LABLOCALE field of the MSAM Label Area.

From the locale identifier, the application can determine the CCS of an image by calling the LOCALE_NUM_TO_CCS_NUM service routine.

2.4.5. Identifying the CCS for an Image Read from a Network

Use of DDP-PPC and IPC$RECEIVE Function

A DDP-PPC application in basic or extended mode uses the IPC$RECEIVE function to receive a message.

Use of IPC-DATA-TYPE Field

The IPC-DATA-TYPE field of the IPC-DATA-PKT is used to indicate what type of data the application would like to receive.

Handling Different Message Types

If the receiving message type is different from the sending type, DDP-PPC transliterates the message.

2.5. Identifying the CCS of a Character When Writing Images

Destination of Data

- To a terminal (see 2.5.1)
- To a system console (see 2.5.2)
- To a file (see 2.5.3)
- To a database (see 2.5.4)
- To a network (see 2.5.5)

2.5.1. Identifying the CCS for an Image Written to a Terminal

If you are operating in	Identify the CCS by using
Basic Mode	ER SYMB$ ER RSI$ SYSLIB SAR$ WRITE SYSLIB SIR$
Extended Mode	SLIB S$SWRITE Compiler language statements Application capabilities

ER SYMB$: Identifying the CCS for an Image Written to a Terminal

Changes to ER SYMB$ Write Function

The ER SYMB$ write function includes 8-bit character set support for the ISO 646 and ISO 8859.*n* coded character sets (CCS).

ER RSI$: Identifying the CCS for an Image Written to a Terminal

Passing CCS Information

On all input to the Exec, coded character set (CCS) information is passed by setting the 010 subfunction bit and placing the character type code in the character set field for the ER RSI$ packet.

Handling Character Set Types

All character types are accepted, excluding the reserved alternate type.

- If the character type is not specified, all RSI$ input is assumed to be ASCII/ISO.
- If the character set designation changes, an 042 SDF image is generated.

SYSLIB SAR$ WRITE: Identifying the CCS for an Image Written to a Terminal or an SDF File/Element

Determining the CCS Identifier and Controlling Return of Images

The CCS identifier of each image to be written is provided by the calling program in the write packet. You can use additional settings in this packet to control which CCSs may be written.

Settings for Write Packet Fields

If the calling program is	The CCS identifier is provided in this field	For these image types	If you set this field	To
Basic Mode PLUS	OUTPUT_IMAGE_CHARACTER_TYPE	All CCSs	UNTRANSLATE_MODE	ON
		Fieldata, ASCII-like, or ACD only		OFF
Basic Mode MASM	SW$OCHART	All CCSs	SW$UNTRFLG	1
		Fieldata, ASCII-like, or ACD only		0

SYSLIB SIR$: Identifying the CCS for an Image Written to a Terminal

Writing Images to a Terminal

If the W option is included when the processor is called, SIR$ is directed to write to the terminal all runstream (terminal) correction images that are read from the terminal. SIR$ uses ER SYMB$ to write all terminal images in the same CCS as they are read.

SLIB S$SWRITE: Identifying the CCS of an Image Written to a Terminal or an SDF File/Element

Determining the CCS Identifier and Controlling Writing of Images

The CCS identifier of each image to be written is provided by the calling program in the write packet. You can use additional settings in this packet to control which CCSs may be written.

Settings for Write Packet Fields

The CCS identifier is returned in this field	As these image types	If you set this field	To
Output_Character_Type	All CCSs	Translate_Mode	Do_Not_Translate
	Fieldata, ASCII-like, ACD, and kanji (Japanese) only		Translate

Compiler Language Statements: Identifying the CCS for an Image Written to a Terminal

Support from COBOL Compiler

The COBOL Compiler support setting a CCS identifier for a file and enable an application to set the locale identifier for a file.

Support from Other Compiler Language Statements

Other compiler language statements set the CCS directly for images written to a terminal. If you are operating in demand mode, you can also use SLIB S$WRITE to access the underlying capabilities.

Application Capabilities: Identifying the CCS for an Image Written to a Terminal

Recognizing Terminal Capabilities

An application running in extended mode must be able to recognize the capabilities of the terminal for which a message is intended.

Generating Protocol Statements

Once the application has determined these capabilities, it can generate the appropriate terminal protocol statements in the message text to indicate the CCS of the message.

Related Documents

For more information about terminal capabilities and appropriate protocols, refer to the documentation for the terminal to which the application is sending its data.

2.5.2. Identifying an Image Written to a System Console

Limited Use of SYSLIB SAR$ COM

This interface allows operator console messages as images in Fieldata, ASCII, and ASCII-like CCSs. ASCII-like CCSs can be output by the calling program, but all are treated as ASCII/ISO images.

2.5.3. Identifying the CCS for an Image Written to a File

If you are operating in	Identify the CCS by using
Basic Mode	SYSLIB SDFO SYSLIB SAR$ WRITE SYSLIB SIR$
Extended Mode	Compiler language statements SLIB S$SWRITE SYSLIB SOR

SYSLIB SDFO: Identifying the CCS for an Image Written to an SDF File/Element

Using the SDF File Control Word Field

Use the SDF Control Word field in the SDFO File Control Table to identify the CCS for each image (data record) to be written.

Using the Label Control Record

The Label Control Record is one of two SDF control record types that can be supplied to determine the CCS identifier of the data records that are to follow.

The Label Control Record

- Is always the first record written.
- Is identified by an octal value of 50 in S1 (bits 0-5) of the SDF Control Word.
- Is the only CCS identifier required if all the data records in the SDF file are in the same CCS.

Using the Character Set Change Control Record

Whenever the CCS identifier changes for the data records in an SDF file, the caller must supply a Character Set Change Control Record before writing data records in the new CCS.

The Character Set Change Control Record

- Is identified by an octal value of 42 in S1 (bits 0-5) of the SDF control word.
- Supplies the CCS identifiers of the data images that follow in S6 (bits 30-35) of the SDF control word.

SYSLIB SIR$: Identifying the CCS for an Image Written to an SDF File/Element

Producing Source Output

SYSLIB SIR$ merges the source input SDF file or element with the runstream correction input to produce an optional source output SDF file or element.

The CCS of the images written to the source output is based on

- The CCS of the source input and runstream images.
- The option characters that are included when the processor is called.

Controlling the CCS of Source Output Images

The P and Q option characters are used in combination to specify the CCS of the source output images. If neither are included, special I-option rules control the CCS of the source output images.

General Rules for P and Q Options

This option	Requests that source output images be written in the
P option	Fieldata CCS
Q option	ASCII-like CCS
P and Q together	Same CCS as the source input and runstream correction images

Special Rules for I Option

If the I option is	The following images	Are written this way
included	Runstream input images	Unchanged to a new source output file or element.
not included	Source input images in all CCSs	Unchanged to the source output file or element.
	Runstream correction images	In the same CCS as the source input image that they are correcting.

Rules for New Runstream Images

New runstream images that follow the	Are written this way
-n correction image	In the same CCS as source input image n.
-n,m correction image	In the same CCS as source input image m.

Compiler Language Statements: Setting the CCS and Locale Identifiers for a File

Compiler Support

The C, COBOL, and Extended Mode PLUS compilers support setting the CCS identifier for a file and enable an I18N application to set the locale identifier for a file.

See also *CODE TYPE Clause on SELECT Statement (COBOL Compiler)* and *LOCALE Clause on SELECT Statement (COBOL Compiler).*

Direct Access

To access the underlying capabilities directly, the application must use SLIB S$SWRITE.

SYSLIB SOR: Identifying the CCS for Images Written to an SDF Element

Producing Symbolic Output

The SOR caller is a processor that has been initialized previously with a call to one of the SYSLIB preprocessor routines. This caller produces symbolic element output rather than omnibus, relocatable, absolute, or object module element output.

Specifying a CCS

Two SOR routines allow the calling program to specify the CCS of each image to be written.

- The SORNM$ routine writes an image with space suppression.
- The SORNMA$ routine writes an image without space suppression.

Writing a CCS

The calling program can write images in all CCSs and places them in Register A1.

2.5.4. Identifying the CCS for an Image Written to a Database

If you are operating in	Identify the CCS by using
Basic or Extended Mode	Network Database Server Relational Database Server PCIOS SFS 2200

Network Database Server: Setting the CCS for a Data Record in a Network Database Server Database File

Use of LOCALE Clause

There is no programmatic way of setting the CCS of a data record in a database file of the Network Database Server. The CCS for all the data under a schema is specified by using the LOCALE clause on the SCHEMA QUALIFIER statement.

PCIOS: Setting the CCS and Locale Identifiers for a PCIOS File

Location of CCS Identifier

The CCS identifier is in S6, the ASCII flag field (also known as the character set field) of the control word for the file header image (SDF 050 image). The values for the CCS identifier match the character set types for SDF files.

Setting the FCFD Field for SSDF and DSDF Files

When an application opens a file for input, it sets the FCFD field in the File Control Block of the File Control Table. PCIOS then writes the resulting ASCII flag value into the label record (SDF 050 image).

FCFD Value	Resulting ASCII Flag Value
1 (Fieldata)	0
0 (ASCII/ISO)	1
2 (ASCII/APL)	2
3 – 77	3 – 77

Writing the EB$LOCALE Identifier for an MSAM File

When an application opens a file for output, it sets the field in the Environment Table to the locale identifier for the data to be stored in the file. PCIOS then writes the EB$LOCALE identifier to the LABLOCALE field of the MSAM Label Area.

SFS 2200: Setting the CCS and Locale Identifiers for an SFS File

Location of CCS Identifier

The CCS identifier is in S6, the ASCII flag field (also known as the character set field) of the control word for the file header image (SDF 050 image). The values for the CCS identifier match the character set types for SDF files.

Setting the FCFD Field for DSDF Files

When an application opens a file for input, it sets the FCFD field in the File Control Block of the File Control Table. SFS then writes the resulting ASCII flag value into the label record (SDF 050 image) and, for the SFS interfaces, sets the EB$CODE field in the environment table to the values shown.

FCFD Value	Resulting ASCII Flag Value
1 (Fieldata)	0
0 (ASCII/ISO)	1
2 (ASCII/APL)	2
3 – 77	3 – 77

Setting the CCS for an MSAM File

When an application opens a file for output, it sets the field in the Environment Table to the locale identifier for the data to be stored in the file. SFS then writes the EB$LOCALE identifier to the LABLOCALE field of the MSAM Label Area.

2.5.5. Identifying the CCS for an Image Written to a Network

If you are operating in basic or extended mode, identify the CCS by using DDP-PPC.

Use of IPC$SEND Function

A DDP-PPC application uses the IPC$SEND function to send a message.

Use of IPC-DATA-TYPE Field

The application indicates in the IPC-DATA-TYPE field of the IPC-DATA-PKT what type of data the application is sending.

Handling Different Message Types

If the receiving message type is different from the sending type, DDP-PPC transliterates the message.

2.6. Using Applications in the I18N Environment

2.6.1. Modifying Application Code for the I18N Environment

Modify this area	So that the I18N application can
8-bit data throughput	Accept 8-bit data instead of rejecting it or stripping off the eighth bit.
8-bit data storage	Use bits 8 and 9 of a quarter word as CCS IDS, not as flags.
Terminal recognition	Recognize and handle new terminals or terminal emulators.
Terminal handlers	Ensure 8-bit character handling.
Printer drivers	Exploit the capabilities of 8-bit printers.
Full character set use	Use 8-bit character sets, rather than filtering characters or converting them to uppercase (if the application stores data as 6-bit characters).

2.6.2. Support for Execution Modes

Applications that use the I18NLIB service routines can operate in these execution modes.

- TIP/HVTIP Mode
- Batch Mode
- Demand Mode
- Real Time Mode

2.6.2.1.TIP/HVTIP Mode Considerations

- Effect of CREATE$BANK Call

  The CREATE$BANK call is used by the I18NLIB service routines to obtain an area for an activity local bank. This bank contains service routine control information. Each activity in a transaction program can have different current locales.

- Storage Allocations for C Compiler SETLOCALE Value

  The SETLOCALE service routine for the C Compiler returns a pointer to a value to the caller. The storage for the value is allocated (1) in the URTS table space in TIP mode and (2) on the HVTIP heap in HVTIP mode.

- Effect of INITIAL Transactions on Category Settings

  Multiple INITAL transactions lose the category settings established by SETLOCALE. At the beginning of the transaction, after INITIAL (the MCB f$init function), the transaction must call SETLOCALE again to reestablish the current settings.

- TIP Session Control and Initial Values for Environment Variables
  - When TIP Session Control is ON, initial values for the environment variables may be present at the run level, depending on the user profile of the user-id.
  - When TIP Session Control is OFF, there are no initial values for the environment variables.
- Effect of SETLOCALE "_DEFAULT" Setting

  If the transaction calls SETLOCALE with "_DEFAULT" before using PUTENV or the system profile mechanism to establish an environment variable value, SETLOCALE sets the specified category to "POSIX" (or to "C" if the application development language is C).

- Environment Variable Settings and System Recovery

  If a transaction is restarted as part of an integrated recovery procedure after a system failure, the environment variable settings are lost. However, no information is lost, since the entire transaction message is re-executed during recovery.

2.6.2.2.Batch Mode Considerations

- Use of CREATE$BANK Call

  The CREATE$BANK call is used by the I18NLIB service routines to obtain an area for an activity local bank. This bank contains service routine control information. Each activity in a batch program can have different current locales.

- Storage Allocation for C Compiler SETLOCALE Value

  The SETLOCALE service routine for the C Compiler returns a pointer to a value to the caller. In batch mode, the storage for the value is allocated in the data bank acquired at the initialization of the C program.

- Setting Initial Values for Environment Variables

  Initial values for the environment variables may be present at the run level, depending on the user profile of the user-id.

- Effect of SETLOCALE "_DEFAULT" Setting

  If the batch mode application calls SETLOCALE with "_DEFAULT" before using PUTENV service routine or the user profile mechanism to establish an environment variable value, SETLOCALE sets the specified category to "POSIX" (or to "C" if the application development language is C).

- Environment Variable Settings and System Recovery

  If a batch session is terminated due to a system failure, @FIN, or other means, the environment variable settings are lost. However, if CKRS is being used, it will reestablish the environment variable values.

2.6.2.3.Demand Mode Considerations

- Use of CREATE$BANK Call

  The CREATE$BANK call is used by the I18NLIB service routines to obtain an area for an activity local bank. This bank contains service routine control information. Each activity in a demand program can have different current locales.

- Storage Allocation for C Compiler SETLOCALE Value

  The SETLOCALE service routine for the C Compiler returns a pointer to a value to the caller. In demand mode, the storage for the value is allocated in the data bank acquired at the initialization of the C program.

- Setting Initial Values for Environment Variables

  Initial values for the environment variables may be present at the run level, depending on the user profile of the user-id.

- Effect of SETLOCALE "_DEFAULT" Setting

  If the demand mode application calls the SETLOCALE service routine with "_DEFAULT" before using the PUTENV service routine or the user profile mechanism to establish an environment variable value, SETLOCALE sets the specified category to "POSIX" (or to "C" if the application development language is C).

- Environment Variable Settings and System Recovery

  If a demand session is terminated due to a system failure, @@TERM, @FIN, $$CLOSE, or other means, the environment variable settings are lost. These values must be reestablished when the next demand session is started.

2.6.2.4.Real Time Mode Considerations

- Effect of CREATE$BANK Call

  The CREATE$BANK call is used by the I18NLIB service routines to obtain an area for an activity local bank. This bank contains service routine control information.

- Environment Variables, Category Settings, and System Recovery

  If a system fails, the recovery of the environment variable and category settings depends on the underlying execution mode, although real-time programs are usually executed in batch mode.

2.7. Testing the New Environment

2.7.1. Testing Techniques for Processing Character Data

Create tests that use searching, comparison, and sorting for a sample of your user's target language.

Sample Tests

If the application	Follow this process	To verify that
Uses the SORT statement or invokes the SORT/MERGE utility	Describe the sort keys	SORT generates culturally sensitive keys.
Uses the Network or Relational Database Server	Use schema statements	Data items described in the schema are processed in a culturally sensitive manner.
Has access to the I18NLIB service routines	Call the service routines directly from the application	Character strings are compared according to culturally sensitive processing rules.

2.7.2. Sample Verification Tests

Create tests to verify each types of processing.

For this type of processing	Create tests for	To verify that
Data processing	Handling characters with eighth bit turned ON for each code type, print file, and symbolic file. Include accepting, storing retrieving, processing, displaying, and printing.	Character-handling occurs with no loss of integrity.
Parsing (general)	Handling different values in the code-type field of both print files and symbolic files, for example,	The application is 8-bit transparent, that is,
Parsing syntax	Creating a 7-bit valid/8-bit invalid syntax.	The application does not ignore the high-order bit; it flags the syntax error.
Parsing comments	Creating a comment with 8-bit data.	The comment is stored and displayed in its 8-bit form, not flagged as an error.

For this type of processing	Create tests for	To verify that
Parsing literals	Creating an 8-bit literal.	The compiled program can process all 8 bits and display the appropriate character.
Parsing words	Create a 7-bit valid/8-bit invalid syntax for key words and reserved words.	The application does not ignore the high-order bit; it flags the word error.
Searching, comparing, and sorting	Searching for, comparing, and sorting characters in an 8-bit binary sequence.	The application uses all 8 bits of a character during processing. Techniques for processing character data

2.8. Using MCB Transliteration Services

The MCB transliteration service helps to migrate a terminal network to 8-bit terminals with minimal impact on existing 7-bit applications. A number of standard conversion tables are provided to assist with transliteration.

2.8.1. Standard Transliteration Tables

Unisys provides these tables to enable an application to translate or transliterate a message from one coded character set (CCS) to another. The system administrator chooses which transliterations to install on a system.

See Appendix B, List of Tables, for the standard transliteration table column names and descriptions and the CSS names and descriptions.

2.8.2. I18N Modifications to TIP and MCB

The TIP COMPOOL-compatible primitives and the relocatable MCB/MCB element used for the MCB packet interface have been modified to carry out character transliteration.

Modifying TIP Primitives

Task: Re-map your transactions to use the special I18N version of these primitives.

Outcome: A transaction can convert 8-bit characters from the ISO 8859.n character sets to the equivalent ISO 646 national replacement character variation on input, and back to the 8-bit equivalents on output.

Modifying the MCB Packet Interface

Task: Remap the MCB/MCB relocatable so that any input or output commands to the MCB will change the text in the data area of the transaction.

Outcome: Input is changed so that the transaction receives the ISO 646 character set. The output buffer is changed so that selected characters are transliterated to the ISO 8859.n characters.

2.8.3. Using @MAP to Collect a Transaction for Transliteration

Use of I18N Utility File

The I18N utility file (I18N$*UTILITY) is used to prepare a transaction for MCB transliteration.

Relocatable Elements Used for TIP Primitives

File I18N$*UTILITY contains different relocatable elements from those found in the standard TIP$*TIPLIB$ file. These include alternative versions of INITAL, CDATAC, CDATCR, CSTLOG, CSTOVR and RTRANU for use with the MCB primitive interface.

Note: *The COMPOOL interface is unaffected. It does not support transliteration.*

Relocatable Elements Used for MCB Packet Users

File I18N$*UTILITY contains the relocatable element MCB/MCB for MCB packet users.

Use of Relocatable Elements

These elements must be used in your transaction only if you want all incoming 8-bit characters to be transliterated to the 7-bit ISO 646 set, and the equivalent characters transliterated back to the 8-bit ISO 8859 on output.

2.8.4. Installing the I18N Utility File

Installation Process

The MCB SOLAR install process creates the I18N$*UTILITY file when you specify the installation mode 'I18N'.

File Functions

This file, which is located on the Communications Delivery release tape, contains utilities to help with transliteration.

Utilities Included

Utilities included are (1) the specialized versions of the TIP primitives that assist with the transliteration, and (2) the relocatable element MCB/MCB for MCB packet users.

2.8.5. Using Conversion Tables

Using Local Conversion Tables

A locally selected conversion table is used to transliterate characters. The standard ISO 646 local variations all have a pre-built table supplied as part of the MCB transliteration feature.

Modifying Local Conversion Tables

You can use the tools supplied with MCB to make additional variations or to meet a requirement for a special purpose local variation.

Mapping Local Conversion Tables

You need to include these tables in the @MAP of the transaction.

Only one table can be mapped into a given transaction, because there is no method to determine which local language variation to choose if more than one is required.

Mapping for Multiple Conversions

If multiple conversions are required, you can map the same transaction using a different program name and transaction code.

2.8.6. Creating Your Own Transliteration Tables

To create your own MCB transliteration tables, follow these steps:

1. Always start with the US symbolic element CONTABUS. This element contains no national replacement characters for the ISO 646 character set.
2. To this version, add a table containing (a) the local replacement characters to the ISO 646 character set and (b) the equivalent character in the ISO 8859 character set.
3. Define local replacement characters. The point of insertion for these replacement characters is clearly marked in the symbolic element CONTABUS.

   Syntax

   ```
   EQUATE ISO646,0XX  ISO8859,0XX
   ```

   where 0*XX* in both cases is the hexadecimal equivalent of the character. The leading 0 is required because it indicates to the MASM compiler that this is a hexadecimal constant.

   Example

   This example defines the only variation for the British ISO 646 character set.

   ```
   EQUATE ISO646,023 ISO8859,0A3
   ```

Description

The ISO 646 character represented by the hexadecimal value '23' (octal 43) is equivalent to the ISO 8859 character represented by the hexadecimal value 'A3' (octal 243).

This means that the standard ISO 646 pound sign character (#) is to be replaced by the British pound sterling sign character (£).

4. Make sure the symbolic country code of the new element contains a name that uniquely identifies this table. A utility program is provided that reads this symbolic element and produces the ISO 8859 and ISO 646 local variations so that you can check their accuracy.
5. After the final symbolic element has been produced, you may use this utility program to print the table.

   Print a Single Table – To print one table at a time, call the following utility program:

   ```
   @I18N$*UTILITY.ISOPRT file.element-name
   ```

   where *element-name* is your new conversion table symbolic element.

   Print Multiple Tables – To print more than one table at a time, add the list of element names to the call line.

   ```
   @I18N$*UTILITY.ISOPRT file.element-name1,file.element-name2,...
   ```

2.8.7. Displaying Transliteration Tables

Hardware Must Expect ISO 8859.n Character Set

The correct characters cannot be displayed unless the following hardware is expecting the ISO 8859.*n* character set:

- The printer that you use to provide a hard copy
- The display terminal upon which the output is presented

Use of '?' Character

Each of the basic 7-bit characters in the ISO 8859.*n* character set contains a '?' character whenever the equivalent character in ISO 646 character set requires transliteration. The '?' indicates that the character cannot be represented after performing output transliteration because it has been converted to the 8-bit equivalent.

2.9. Security Considerations

2.9.1. Security Attributes

I18NLIB Security Environment

I18NLIB operates in a Security Level 3 Trusted Environment.

Protection Mechanisms

Once a conversion is made, the I18N database files and the residual data in memory are protected from unauthorized access by a combination of subsystem characteristics and addressing techniques.

Subsystem Characteristics: Use of Chameleon Subsystem

I18NLIB executes as a chameleon subsystem. A chameleon subsystem attaches itself to whatever the caller is and inherits the security attributes of the calling program. Therefore, the only data that can be accessed is that which is accessible to the caller.

In a chameleon subsystem, the data bank is owned by the home system of the calling application. (In a standalone subsystem, the databank is owned by the subsystem itself.)

Addressing Techniques: Use of Exec ASTORE Feature

I18NLIB uses the Exec Activity Local Storage (ASTORE) feature to store the virtual address (VA) of the data bank created by CREATE$BANK. Because ASTORE works at an activity level, both the ASTORE data and the data bank are no longer accessible when the activity terminates. Therefore, there is no residual data to protect.

2.9.2. TIP Security

Security Implications of TIP Session Control ON

If TIP Session Control is ON, individual user-ids are used to access the system at the run level. From the user profile, you can initialize the environment variables available through TIP. These variables can then be used for the SETLOCALE default process.

Note: *TIP Session Control ON provides a more secure environment, but processing is slower because of the additional checking required.*

Security Implications of TIP Session Control OFF

If TIP Session Control is OFF, the system is accessed at the system level. Since no run-level information is available, you must either (1) use the system environment variables, or (2) initialize the environment variables programmatically within each transaction.

Note: *TIP Session Control OFF is less secure, but processing is faster.*

2.10. Recovery Issues

Checkpoint/Restart Not Supported

The I18N Service Library (I18NLIB) does not support CKRS, so any batch runs that were executing when the system failed must be restarted from the beginning. I18NLIB service routines detect that the batch program has been restarted and return an error status.

Category Values Not Recoverable

The category values in use before a system failure are not recoverable. Any transaction, batch, or demand program that is restarted has no knowledge of previous category settings. As with any program initiation, the default category values are re-established.

2.11. I18NLIB Error Processing

Error Structure

The error structure for the I18NLIB service routines uses the constructs of the individual language bindings.

Error Statuses

All I18NLIB service routines except for the C language bindings use a common set of error statuses and error status definitions. These common definitions are listed in Appendix A. The C language error statuses are listed in the C language section (Section 3). The error statuses that each service routine binding can return for a language are listed with the description of the binding for that language.

Severity Codes for Error Statuses

0	SUCCESS
2	INFORMATION
3	WARNING
7	FAILURE

2.12. Detecting, Displaying, and Verifying Errors

Detecting

The I18NLIB service routines return an ELMS condition identifier for any detected errors to the caller.

Displaying

The calling program passes this identifier to ELMS which then displays the diagnostic message. For the C language, *errno* is set to the error code and the *perror*() function can be used to print the error message.

Verifying

The I18NLIB service routines are not required to do parameter error checking. However, the number of parameters passed from language calls are verified and an appropriate status is returned (except for calls from Basic Mode PLUS, Extended Mode PLUS, and the C Compiler).

Assumptions

For extended mode MASM calls, it is assumed that X10 contains a pointer to the ALS.

Section 3
C Compiler

This section describes the requirements to use I18NLIB functionality in the C language. It contains the requirements to access the I18NLIB versions of the C functions and describes the I18NLIB versions of the C functions.

3.1. C Requirements

3.1.1. Keyword Options for Accessing I18NLIB Service Routines and Enabling I18N Processing

Enabling I18N Processing in New C Compiler Applications

To enable I18N processing in these applications, specify the I18N/I18NLIB keyword option on the C Compiler call line.

Enabling I18N Processing in Earlier C Compiler Applications

With the introduction of I18N processing, several features have been added to preserve compatibility with existing applications that use the C Compiler.

Using I18NLIB With the C Compiler

The C Compiler uses the service routines in the I18N Service Library (I18NLIB) to provide internationalized (I18N) processing for applications. These service routines are alternate versions to the run-time library previously released with the C Compiler.

Using Sort/Merge Interfaces

Data records can be sorted or merged in a culturally sensitive manner by using the Sort/Merge C Compiler linkages. These linkages are described in the *Sort/Merge Programming Guide* (7831 0687).

3.1.2. Making I18N Features Compatible With Application Programs

Processing Earlier C Compiler Application Programs

In the C Compiler runtime system, the standard C functions *strcoll*() and *strxfrm*(), as well as the ISA functions *isalpha*() and so on, operate strictly by the US English rules for the ASCII code set.

In the I18N environment, any existing C Compiler programs that use these functions are still processed according to these same rules.

Processing Existing C Compiler Applications in the I18N Environment

With the introduction of I18N processing, there are the following ways to deal with existing application programs:

- Leave them alone.

  Any calls to *setlocale*(), *strcoll*(), *strxfrm*(), and the ISA functions operate as they have in the past.

- Direct the compiler to access the I18NLIB service routines.

  If you include the I18N/I18NLIB keyword option on the compiler call line, the C compiler accesses the I18NLIB versions of the service routines rather than the ASCII versions.

- Direct specific application calls to the I18NLIB.

  If you change only certain application calls to *strcoll*() and *strxfrm*() to be calls directly to I18NLIB, the I18NLIB exports the function names preceded by two underscore characters, as in *__setlocale*(), *__strcoll*(), *__strxfrm*(), *__isalpha*(), and so on. The underscore characters distinguish these names from the standard function names.

3.1.3. Enabling I18N Features in Application Programs

Turning On I18N Processing

To preserve compatibility in applications created with previous levels of the C Compiler, you must explicitly turn on I18N processing when you compile your application program. The new keyword options to include on the C Compiler call line are

```
I18N/I18NLIB
I18N/CLIB
```

where

```
I18N/I18NLIB
```

is the keyword option that enables I18N processing.

```
I18N/CLIB
```

is the keyword option that preserves compatibility with previous levels of the compiler. This is the default setting.

Using the I18N/I18NLIB Keyword Option

Including the I18N/I18NLIB keyword option on the call line changes the way that the C Compiler handles any calls to *setlocale*(), *strcoll*(), *strxfrm*(), or to the ISA functions such as *isalpha*() and *isalnum*().

These references are now resolved to the I18N version of the service routines instead of to the runtime library that supports the ASCII character set exclusively and applies US English rules only.

Processing with the I18NLIB Service Routines

I18NLIB processes the *setlocale*(), *strcoll*(), *strxfrm*(), and ISA functions based on the current locale settings.

Only those service routines described in C Compiler service routines are internationalized. All others continue to function strictly with ASCII processing rules.

Performing Comparisons

If the standard C comparison operators (such as > < == <>) are used, a binary comparison is done on the data items.

To perform a culturally sensitive comparison, the application has to call either the *strcoll*() or the *strxfrm*() function.

3.1.4. C Compiler Calls: Sample Code for Calling the I18N Service Library

The following examples show the various options for calling the I18N service library (I18NLIB). You can use your favorite text editor to copy and paste them into your applications.

- Example 1: Compiler call with no options or the I18N/CLIB option for compatibility
- Example 2: Compiler call with no options or the I18N/CLIB option and calling the I18NLIB routines directly
- Example 3: Compiler call with the I18N/I18NLIB option to have the compiler invoke the I18NLIB routines

3.1.4.1. Example 1: C Compiler Call With No Options or I18N/CLIB Option

This code example shows how to call the C compiler with either no options or with the I18N/CLIB option.

```
@ . Call the C Compiler with no options.
@ .
@UC,I infile.elt1,outfile.elt1
@ .
@ . This is equivalent to
@ .
@UC,I infile.elt1,outfile.elt1,,,I18N/CLIB
main()
{
#include <locale.h>
#include <string.h>

char c1[5] = "abcdë";
char c2[5] = "EFGHÏ";
```

```
/* The following statement returns TRUE because C compares
characters based on their binary codes.  In ASCII, all uppercase
characters precede all lowercase characters.  */

if (c1 > c2)
     { printf("EFGHÏ comes before abcdë in ASCII");
     }

/* The following statement returns TRUE because the C library
strcoll() function compares characters based on their binary
codes.  In ASCII all uppercase characters precede all lowercase
characters.  */

if (strcoll(c1,c2) < 0)
     { printf("EFGHÏ comes before abcdë in ASCII");
     }

/* The following statement returns FALSE because the
diacritical letter ë is not part of the ASCII character set.  */

if (isalpha(c1[5]))
     { printf("abcdë contains all alphabetic characters");
     }
}
@EOF
```

3.1.4.2. Example 2: C Compiler Call With No Options, I18N/CLIB Option, Direct Call to I18NLIB

This code example shows how to call the C compiler with either no options, with the I18N/CLIB option, or by a direct call to the I18N Service Library.

```
@ . Call the C Compiler with no options and call the I18NLIB
@ . routines directly.
@ .
@UC,I infile.elt1,outfile.elt1
@ .
@ . This is equivalent to
@ .
@UC,I infile.elt1,outfile.elt1,,,I18N/CLIB
main()
{
#include <locale.h>
#include <string.h>

char c1[5] = "abcdë";
char c2[5] = "EFGHÏ";
char *locale;
```

```
/* Establish the locale for the I18NLIB routines to use.
Using the __setlocale() calling sequence invokes the I18NLIB
routine directly.  */

locale = __setlocale(LC_ALL,"fr_FR.8859-1");

/* The following statement returns TRUE because C compares
characters based on their binary codes.  In ASCII all uppercase
characters precede all lowercase characters.  */

if (c1 > c2)
     { printf("EFGHÏ comes before abcdë in ASCII");
     }

/* The following statement returns TRUE because the I18NLIB
strcoll() function performs a culturally-sensitive comparison
based on the French language processing rules for France using
the ISO 8859.1 character set (fr_FR.8859-1).  The characters
are first changed to the same case.  Thus, A comes before E.
Using the __strcoll() calling sequence invokes the I18NLIB
routine directly.  */

if (__strcoll(c1,c2) < 0)
     { printf("abcdë comes before EFGHÏ in French");
     }

/* The following statement returns FALSE because the
diacritical letter ë is not part of the ASCII character set.
Note that since the function call is not preceded by two
underscore characters (__isalpha), the C library isalpha()
function is invoked, not I18NLIB isalpha() function.  */

if (isalpha(c1[5]))
     { printf("abcdë contains all alphabetic characters");
     }
}
@EOF
```

3.1.4.3.Example 3: C Compiler Call With I18N/I18NLIB Option

This code example shows how to call the C compiler by using the I18N/I18NLIB option.

```
@ . Call the C Compiler with the I18N/I18NLIB option.
@ .
@UC,I infile.elt1,outfile.elt1,,,I18N/I18NLIB
main()
{
#include <locale.h>
#include <string.h>
```

```
char c1[5] = "abcdë";
char c2[5] = "EFGHÏ";
char *locale;

/* Establish the locale for the I18NLIB routines to use.
Using the setlocale() calling sequence along with the
I18N/I18NLIB compiler keyword invokes the I18NLIB
routine.  */

locale = setlocale(LC_ALL,"fr_FR.8859-1");

/* The following statement returns TRUE because C compares
characters based on their binary codes.  In ASCII all uppercase
characters precede all lowercase characters.  All comparisons
using the standard comparison operators still use a binary
comparison, even with the I18N/I18NLIB keyword.  */

if (c1 > c2)
     { printf("EFGHÏ comes before abcdë in ASCII");
     }

/* The following statement returns TRUE because the
I18NLIB strcoll() function performs a culturally-sensitive
comparison based on the French language processing rules
for France using the ISO 8859.1 character set (fr_FR.8859-1).
The characters are first changed to the same case.  Thus,
A comes before E.  Using the strcoll() calling sequence along
with the I18N/I18NLIB compiler keyword invokes the I18NLIB
routine.  */

if (strcoll(c1,c2) < 0)
     { printf("abcdë comes before EFGHÏ in French");
     }

/* The following statement returns TRUE because the
diacritical letter ë is an alphabetic character in the
fr_FR.8859-1 locale.  The I18N/I18NLIB keyword causes
the I18NLIB function to be invoked.  */

if (isalpha(c1[5]))
     { printf("abcdë contains all alphabetic characters");
     }
}
@EOF
```

3.2. C Error Structure

C Compiler programs can use the declarations that are found in the <errno.h> standard include file.

Defining the Value for *errno*

On a normal return, the value for the *errno* global variable is undefined. The user program should set this value to zero before calling a service routine. If an error is detected by a service routine, *errno* is set to indicate the error.

For information on how the caller can detect an error status, refer to the C Compiler language binding for each service routine.

Printing an Error Message

The *perror*() function may be called to print an error message. The message is printed in the format

```
*ERROR (I18NLIB) 130nn: message-text
```

Error Declarations: Header File errno.h

The following error statuses defined in the *errno.h* header file can be returned by the C Compiler service routines. For an expanded description of each error, see 3.3.

```
#define _E_ISL_INVALID_PACKET_VERSION 13002
  /* Invalid packet version for NAME_AND_NUMBER packet interface */

#define _E_ISL_2_SHORT 13003
  /* The supplied buffer is too short */

#define _E_ISL_BAD_FROM 13004
  /* The from-ccs transliteration table not available */

#define _E_ISL_BAD_TO 13005
  /* The to-ccs transliteration table not available */

#define _E_ISL_CHAR_ENCODING_BAD 13007
  /* A character with an encoding not defined in the from-ccs codes
  encountered */

#define _E_ISL_NOT_IN_CCS 13013
  /* The arguments contain characters outside domain of collating
  sequence */

#define _E_ISL_BAD_CD 13014
  /* The conversion descriptor is invalid */

#define _E_ISL_TOO_MANY_PARAMETERS 13015
  /* There were too many parameters passed to the service routine */

#define _E_ISL_TOO_FEW_PARAMETERS 13016
  /* There were too few parameters passed to the service routine */
```

```
#define _E_ISL_BADCHAR 13020
  /* An invalid character was found in the environment variable name */

#define _E_ISL_VAL_BADCHAR 13021
  /* An invalid character was found in the environment variable
  value */

#define _E_ISL_ENV_AREA_FULL 13022
  /* The Exec-defined area in the PCT to store environment variables
  and their values is full. The specified name and value cannot be
  stored */

#define _E_ISL_BAD_TIP_EV 13023
  /* Specified environment variable not allowed in TIP */

#define _E_ISL_NAME_NOT_FOUND 13024
  /* The specified environment variable name was not found */

#define _E_ISL_VAL_2_LONG 13025
  /* The environment variable value is longer than the maximum
  length allowed in TIP */

#define _E_ISL_LOCALE_NOT_FOUND 13026
  /*The specified locale cannot be found */

#define _E_ISL_BAD_CATEGORY 13027
  /* An invalid category was specified */

#define _E_ISL_CORRUPT_TABLE 13030
  /* The table used by the service routine has been corrupted */

#define _E_ISL_CCS_NOT_FOUND 13031
  /* The specified CCS cannot be found */

#define _E_ISL_CCS_THE_SAME 13032
  /* In a call to OPEN_CCS_TO_CCS, the from-ccs and to-ccs codes are
  the same */

#define _E_ISL_CCS_BANK_FULL 13033
  /* No more OPEN_CCS_TO_CCS calls can be made, the data bank is
  full */

#define _E_ISL_STRING1_2_LONG 13036
  /* The first string passed to the service routine is too long */

#define _E_ISL_STRING2_2_LONG 13037
  /* The second string passed to the service routine is too long */

#define _E_ISL_PUT$ENV_ERROR 13038
  /* An error not defined by I18NLIB was returned by the Exec */
```

```
#define _E_ISL_GET$ENV_ERROR 13039
  /* An error not defined by I18NLIB was returned by the Exec */

#define _E_ISL_CCS_NOT_UNIQUE 13040
  /* CCS SDF/ISO Number passed to CCS_NUM_TO_CCS_NAME is not unique */

#define _E_ISL_ASTORE_READ 13041
  /* An error was encountered while performing an ASTORE read */

#define _E_ISL_ASTORE_WRITE 13042
  /* An error was encountered while performing an ASTORE write */

#define _E_ISL_CREATE_BANK 13043
  /* An error was encountered while performing a create bank */

#define _E_ISL_CKRS_ERROR 13044
  /* A checkpoint restart was encountered after an ASTORE read */

#define _E_ISL_CCS_FORM_BAD 13045
  /* The specified CCS format is invalid */

#define _E_ISL_BAD_ITEM 13050
  /* Invalid locale item passed to nl_langinfo */

#define _E_ISL_CONV_SPEC 13051
  /* Invalid conversion specification used in STRFMON, STRFTIME,
  or STRPTIME call */

#define _E_ISL_BAD_ARGS 13052
  /* Invalid argument passed to STRFMON */

#define _E_ISL_FEW_ARGS 13053
  /* Fewer arguments than are indicated by the format str were passed
  to strfmon() */

#define _E_ISL_CHR_MISMATCH 13054
  /* STRPTIME encountered a mismatch in characters between the input
  & format strings */

#define _E_ISL_MODIFIED_CS 13055
  /* The modified conversion specification used in an STRFTIME or
  STRPTIME format string is not supported by I18NLIB */

#define _E_ISL_INVALID_DATE_TIME 13056
  /* Invalid Date/Time value for STRFTIME or STRPTIME call */

#define _E_ISL_INVALID_NMNB_ID 13057
  /* Invalid packet identifier for NAME_AND_NUMBER packet interface */

#define _E_ISL_NMNB_NAME_LENGTH 13058
  /* Invalid name length in NAME_AND_NUMBER packet */
```

```
#define _E_ISL_SYSTEM$TIME_ERROR 13059
  /* An error was returned by the Exec for the SYSTEM$TIME call. */
```

3.3. Error Codes

This section describes in detail the errors detected and returned by the I18NLIB C functions in the *errno* global variable.

Error Code 13001: Invalid Packet Length

The value supplied is not a valid length for the service routine packet.

Change the value to a valid length.

Error Code 13002: Invalid Packet Version

The value in the version field is not valid.

For the NAME_AND_NUMBER packet interface, this value must be 1.

For all other service routines, this value must be 0 or 1.

Error Code 13003: Buffer Too Short

The supplied buffer pointed to by one of the output parameters is too short to hold the resultant value being returned.

Increase the length of the output data item buffer.

Error Code 13004: FROM-CCS Not Defined

The transliteration table is not available for the CCS pointed to by *from_css*. The CCS cannot be found on the system.

If you cannot verify the CCS number, contact your system administrator. The CCS might not be installed on your system.

Error Code 13005: TO-CCS Not Defined

The transliteration table is not available for the CCS pointed to by *to_ccs*. The CCS cannot be found on the system.

If you cannot verify the CCS number, contact your system administrator. The CCS might not be installed on your system.

Error Code 13007: Character Encoding Unknown

A character in the string pointed to by *s1* or *s2* is unknown in LC_COLLATE category associated with the current locale.

Change either the locale associated with LC_COLLATE category or correct the character in the string pointed to by *s1* or *s2* so that the character and locale combination is valid.

Error Code 13013: Bad Character

- Input conversion stopped because a character was encountered that does not belong to the input codeset.

  The *inbuf* pointer points to the invalid character. Correct this character and call CCS_TO_CCS again.

- A character was encountered that is outside the domain of the collating sequence.

  Contact your system administrator. The CCS may not be installed on your system.

Error Code 13014: Bad Descriptor

The conversion descriptor indicated by *cd* is not valid.

Make sure that the value in *cd* is the one returned by OPEN_CCS_TO_CCS.

Error Code 13015: Too Many Parameters

There are too many parameters in the calling sequence, declaration, or parameter list.

Remove the incorrect parameters.

Error Code 13016: Too Few Parameters

There are too few parameters in the calling sequence, declaration, or parameter list.

Add the missing parameters.

Error Code 13020: Environment Variable Name Invalid Character

- The EV name in the string pointed to by name contains a '=', NULL, space, or control character, each of which is illegal for an EV name.

  Change the illegal character to a printable character. The character codes are always assumed to be from the ISO 8859.1 character set.

- The name portion in the string pointed to by string contains a '=', NULL, space, or control character, each of which is illegal for an EV name.

  Change the illegal character to a printable character. The character codes are always assumed to be from the ISO 8859.1 character set.

Error Code 13021: Environment Variable Value Invalid Character

The value portion in the string pointed to by string contains a '=', NULL, space, or control character, each of which is illegal for an EV value.

Change the illegal character to a printable character. The character codes are always assumed to be from the ISO 8859.1 character set.

Error Code 13022: Environment Variable Area Full

The data space in the run control tables allocated for environment variables and their values is full. In a demand or batch run, this space can contain 4096 characters, including the EV names, the '=' separators, and the EV values.

Once environment variables are defined, the only way to remove them is to restart the run, which clears out the data space.

Error Code 13023: Environment Variable Name Illegal for TIP

In the TIP environment, only LANG, LC_ALL, LC_CTYPE, LC_COLLATE, LS_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME can be used as EV names.

Make sure that these names are the only ones used in the application.

Error Code 13024: Environment Variable Name Not Found

The EV name in the string pointed to by name cannot be found in the run-level or system-level set of environment variables.

If you cannot verify the EV name, use PUTENV to create this variable, GETENVSYS to search for it at the system level, or GETENV to search for it at the run-level.

Error Code 13025: Environment Variable Value Too Long

The EV value cannot exceed 16-characters. In the TIP environment, the value of LANG cannot exceed 10 characters.

Change the value to stay within the character limit.

Error Code 13026: Locale Not Found

The locale name specified in the string pointed to by locale cannot be found on the system.

Locale names are case-sensitive. If you cannot verify the spelling of this name, contact your system administrator. The locale may not be installed on your system.

Error Code 13027: Not a Unisys Intermediate Locale

The locale identified by the LC_CTYPE category is not a Unisys intermediate locale and does not contain the rules necessary to perform this service routine.

All Unisys intermediate locales have "MIG" associated with the CCS name (for example, de_DE.MIG646DE). The system administrator can install these locales on your system.

Error Code 13028: Invalid Category

The value indicated for the category is not valid.

Make sure that the category contains one of these values:

0 = LC_ALL
1 = LC_COLLATE
2 = LC_CTYPE
3 = LC_MESSAGES
4 = LC_MONETARY
5 = LC_NUMERIC
6 = LC_TIME

Error Code 13029: Category Not Set

The LC_CTYPE category does not have a value.

You need to use the SETLOCALE service routine to set LC_CTYPE before you call the CONVERT_FORM service routine.

Error Code 13030: Corrupted Table

The data bank containing locale data items, locale definitions, or CCS-TO-CCS translation information has been corrupted or the bank contains incorrect versions of individual tables.

Reload the AFCB data bank. The BDI for this bank is 0405234.

If you reload the bank while applications are accessing it, they will abort.

If the bank reload fails, have your system administrator reinstall the locales and the CCS-TO-CCS conversions on your system.

If the bank contains incorrect versions of individual tables, then the bank must be rebuilt using the updating and installing runtime tables process. This situation can occur if I18NLIB level 2R1 or higher is installed and the runtime tables were created using an I18NLIB level lower than 2R1.

Error Code 13031: CCS Not Found

The CCS name specified in the string pointed to by *ccsnam* cannot be found on the system.

CCS names are case-sensitive. If you cannot verify the spelling of this name, contact your system administrator. The CCS may not be installed on your system.

Error Code 13032: FROM CCS Equals TO CCS

The CCSs are the same in both the *from_ccs* and the *to_ccs* fields. Therefore, the transaction is invalid.

(1) Ignore the error. Skip the CCS_TO_CCS and CLOSE_CCS_TO_CCS service routines and continue processing, or

(2) Change the application so that the *from-ccs* and *to-ccs* fields contain different CCS identifiers.

Error Code 13033: CCS Bank Full

The data bank containing the CCS transliteration tables is full. No more tables can be added.

Contact your Unisys representative. This situation should not occur unless you have referenced more than 30,000 transliteration tables.

Error Code 13036: String 1 Too Long

The string pointed to by *s1* is more than 4096 characters long.

Change the application to pass the information in this string to the service routine in 4096 character pieces.

Error Code 13037: String 2 Too Long

The string pointed to by *s2* is more than 4096 characters long.

Change the application to pass the information in this string to the service routine in 4096 character pieces.

Error Code 13038: PUT$ENV Error

During a call to the operating system service routine PUT$ENV, the Exec has returned an error not defined by I18NLIB. No other information is available.

Retry the application. If the failure is persistent, contact your Unisys representative.

Error Code 13039: GET$ENV Error

During a call to the operating system service routine GET$ENV, the Exec has returned an error not defined by I18NLIB. No other information is available.

Retry the application. If the failure is persistent, contact your Unisys representative.

Error Code 13040: CCS Number Not Unique

The CCS SDF/ISO number passed to this service routine is not unique. Therefore, a CCS name cannot be returned.

The Unisys CCS number is what makes the CCS unique; use this number to identify the CCS.

Error Code 13041: ASTORE Read Error

The system was unable to read the subsystem ASTORE word. This word is used to obtain the virtual address (VA) of the data bank that contains category information for a program.

Contact your system administrator. Restart your program after the problem has been resolved.

Error Code 13042: ASTORE Write Error

The system was unable to write the subsystem ASTORE word. This word is used to obtain the virtual address (VA) of the data bank that contains category information for a program.

Contact your system administrator. Restart your program after the problem has been resolved.

Error Code 13043: Create Bank Error

The system was unable to create the data bank used to store category information for a program.

Contact your system administrator. Restart your program after the problem has been resolved.

Error Code 13044: Checkpoint Restart Error

After recovering from a system crash, the program restarted at an internal checkpoint, not at the beginning. The system crash invalidated the program's category bank, so all category data for that program was lost.

Restart the program to recreate the category bank.

Error Code 13045: CCS Number Format Undefined

The CCS number format passed to the service routine is not valid.

Please correct the CCS number format to a valid value.

Error Code 13050: Invalid Locale Item

A call to the NL_LANGINFO service routine specified an invalid locale item.

Change the NL_LANGINFO call to specify a valid locale item. Valid locale items for the C Compiler are located in langinfo.h.

Error Code 13051: Invalid Conversion Specification

An invalid conversion specification was passed in a format string to either the STRFMON, STRFTIME, or STRPTIME service routine.

Change the service routine call to specify a valid conversion specification. For a list of valid conversion specifications, go to the service routine description for the language binding that you are using.

Error Code 13052: Invalid Argument Value

The argument passed to the STRFMON service routine does not follow a recognized floating point number format.

Change the number of arguments to match the number of conversion characters in the STRFMON format string.

Error Code 13053: Too Few Arguments

Fewer arguments were passed to the STRFMON service routine than are indicated by the format string.

Increase the number of arguments to correspond to the number of conversion specifications in the format string.

Error Code 13054: Descriptor Does Not Match Input Chr

The STRPTIME service routine encountered an inconsistency while parsing the input and format strings.

Change the in-string character and the corresponding format-string character so that they are equal to each other.

Error Code 13055: Modified Conversion Specification Not Supported

The I18NLIB service routines do not support the modified conversion specification (CS) used in an STRFTIME or STRPTIME format string.

Change the modified CS to a valid unmodified CS. For more details about conversion specifications, see the Conversion Specification Table.

Error Code 13056: Invalid Date/Time Value

There is an invalid date/time value in the STRFTIME or STRPTIME call.

Change the call to specify a valid date/time value. Valid date/time values are specified in langinfo.h.

Error Code 13057: Invalid NMNB Packet Identifier

There is an invalid packet identifier in the call to the NAME_AND_NUMBER packet interface.

Change the call to specify a valid packet identifier. Valid values are specified in locale.h.

Error Code 13058: Invalid Name Length in NMNB Packet

The name length in the NAME_AND_NUMBER packet contains an invalid value.

Change the call to specify a valid value for the CCS or locale name.

Error Code 13059: SYSTEM$TIME Error

The I18NLIB SRTFTIME service routine called the Exec call SYSTEM$TIME and the Exec returned an unexpected error status to I18NLIB.

This is either an internal error or the levels of the Exec and I18NLIB are not compatible. If the levels of I18NLIB and the Exec are not from the same or a compatible release, then please install compatible levels of I18NLIB and the Exec. If the levels of the Exec and I18NLIB are compatible, then this is an internal error and a UCF should be submitted.

3.4. Date/Time Structures (STRFTIME and STRPTIME)

The date/time structure for these two service routines is a nine-word integer packet.

Table 3–1. STRFTIME and STRPTIME Date/Time Structures

This word	Contains or indicates	And must be within this range
0	Number of seconds after the minute. The upper limit is 60 for handling "leap seconds."	[0, 60]
1	Number of minutes after the hour.	[0, 59]
2	Number of hours since midnight.	[0, 23]
3	Day of the month.	[1, 31]
4	Number of months since January. If the current month is January, this number is 0.	[0, 11]
5	Number of years since 1900 A -1 in this field indicates the year 1899. A 100 in this word indicates the year 2000.	[-1899, 8099]
6	Number of days since Sunday. If the current day is Sunday, this number is 0.	[0, 6]
7	Number of days since January 1. If the current day is January 1, this number is 0.	[0,365]
8	Status of Daylight Savings Time positive, DST in effect; 0, DST not in effect, and negative, no DST information available	language-specific

3.5. ccs_name_to_ccs_num() Function

Name

cnm2cnb()

Description

ccs_name_to_ccs_num() transforms a coded character set (CCS) name to its corresponding CCS number.

Declarations

```
#include <locale.h>
int cnm2cnb(char *ccsnam, int ccsformat);
```

Input Values

<locale.h>

is the run-time library function call for locale information. This is the standard header in which the corresponding CCS name is declared.

ccsnam

is a pointer to the string containing the CCS name. This name is case-sensitive and must correspond to a CCS_NAME entity in the Repository and a CCS that is active on the system.

ccsformat

is the kind of CCS number desired. It can be in the FMTCCS, FMTISO, or FMTSDF formats.

FMTCCS is the Unisys CCS number for use with *ccs_to_ccs*() transliteration.

FMTISO is the ISO CCS number for interoperability use.

FMTSDF is the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *ccs_name_to_ccs_num*().

- If completed successfully, this function returns the CCS number, a numeric code corresponding to the specified CCS name.
- If an error occurs, the function sets *errno* and returns a -1.

Error Statuses

ccs_name_to_ccs_num() can return the following error statuses in the *errno* global variable:

- _E_ISL_CCS_NOT_FOUND
- _E_ISL_CCS_FORM_BAD

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *ccs_name_to_ccs_num*() to transform a CCS name to its corresponding CCS number.

```
int ccsnum;
ccsnum = cnm2cnb("ISO646DE", FMTSDF);
```

3.6. ccs_num_to_ccs_name() Function

Name

cnb2cnm()

Description

ccs_num_to_ccs_name() converts a coded character set (CCS) number to its corresponding CCS name.

Declarations

```
#include <locale.h>
void cnb2cnm(int ccsformat, int ccsnum, char *ccsnam, size_t n);
```

Input Values

<locale.h>

is the run-time library function call for locale information. This is the standard header in which the corresponding CCS number is declared.

ccsformat

is the kind of CCS number provided. It can be in the FMTCCS, FMTISO, or FMTSDF formats.

FMTCCS is the Unisys CCS number for use with *ccs_to_ccs*() transliteration.

FMTISO is the ISO CCS number for interoperability use.

FMTSDF is the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records.

ccsnum

is a CCS number.

ccsnam

is a pointer to the resulting CCS name.

n

is the number of bytes that can be placed in the resulting CCS name, including the terminating null byte.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *ccs_num_to_ccs_name*().

- If completed successfully, this function returns the CCS name corresponding to the specified CCS number.
- If an error occurs, the function sets *errno* and the value for *ccsnam* is undefined.

Error Statuses

ccs_num_to_ccs_name() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_CCS_NOT_FOUND
- _E_ISL_CCS_NUM_NOT_UNIQUE
- _E_ISL_CCS_FORM_BAD

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *ccs_num_to_ccs_name*() to transform a CCS number to its corresponding CCS name.

```
char *cnm;
cnb2cnm(FMTSDF, 35, cnm, 10);
```

3.7. ccs_to_ccs() Function

Names

iconv()

Description

ccs_to_ccs() is one of the three service routines used to transliterate a character string from one coded character set (CCS).

Declarations

```
#include <iconv.h>
iconv_t iconv_open(const char *tocode, const char *fromcode);

size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft,
char **outbuf, size_t *outbytesleft);

int iconv_close(iconv_t cd);
```

Input Values

<iconv.h>

is the run-time library function call for character conversion. This is the standard header in which the three string transliteration functions are declared.

tocode

is a pointer to a string that identifies the CCS resulting from the transliteration. This is a decimal number.

fromcode

is a pointer to a string that identifies the CCS to be transliterated. This is a decimal number.

cd

is the conversion descriptor (internal identifier for the transliteration tables) returned by *open_ccs_to_ccs*().

inbuf

is a pointer to a variable that points to the first character in the input buffer.

inbytesleft

is an unsigned integer that indicates the number of bytes (Fieldata characters) to the end of the buffer to be converted.

outbuf

is a pointer to a variable that points to the first available byte in the output buffer.

outbytesleft

is an unsigned integer that indicates the number of available bytes (Fieldata characters) to the end of the buffer.

Return Values

- If completed successfully, this function updates the variables pointed to by the arguments to reflect the extent of the conversion, and returns the number of non-identical conversions performed.
- If the entire string in the input buffer is converted, the value pointed to by *inbytesleft* is zero (0).
- If the input conversion is stopped, the value pointed to by *inbytesleft* is non-zero, and *errno* is set to indicate the condition.
- If an error occurs, *ccs_to_ccs*() returns a -1 and sets *errno* to indicate the error.

Error Statuses

The string transliteration service routines can return the following error statuses in the error field:

- _E_ISL_2_SHORT
- _E_ISL_BAD_FROM

- _E_ISL_BAD_TO
- _E_ISL_NOT_IN_CCS
- _E_ISL_BAD_CD
- _E_ISL_CORRUPT_TABLE
- _E_ISL_CCS_NOT_FOUND
- _E_ISL_CCS_THE_SAME
- _E_ISL_CCS_BANK_FULL

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *open_ccs_to_ccs*(), *ccs_to_ccs*(), and *close_ccs_to_ccs*() to transliterate a character string from one CCS (ISO 646ES) to another (ISO 8859.1).

```
#include <iconv.h>
       .
       .
       .
int i;
iconv_t  cd;
    .
    .
    .
/* The ISO 646ES characters are in the buffer pointed to */
/* by inbuf_ptr, and the converted characters are placed */
/* in the buffer pointed to by outbuf_ptr.               */

cd = iconv_open("23","35");
if (cd == -1)
     printf("Conversion not available.\n");
else
  {
  i = iconv(cd,&inbuf_ptr,&inbuf_size,&outbuf_ptr,&outbuf_size);
  if (i == -1)
     printf("Conversion failed.\n");
  iconv_close(cd);
  }
    .
    .
    .
```

3.8. close_ccs_to_ccs() Function

Names

iconv_close()

Description

close_ccs_to_ccs() is one of the three service routines used to transliterate a character string from one coded character set (CCS).

Declarations

```
#include <iconv.h>
iconv_t iconv_open(const char *tocode, const char *fromcode);

size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft,
char **outbuf, size_t *outbytesleft);

int iconv_close(iconv_t cd);
```

Input Values

<iconv.h>

is the run-time library function call for character conversion. This is the standard header in which the three string transliteration functions are declared.

tocode

is a pointer to a string that identifies the CCS resulting from the transliteration. This is a decimal number.

fromcode

is a pointer to a string that identifies the CCS to be transliterated. This is a decimal number.

cd

is the conversion descriptor (internal identifier for the transliteration tables) returned by *open_ccs_to_ccs*().

inbuf

is a pointer to a variable that points to the first character in the input buffer.

inbytesleft

is an unsigned integer that indicates the number of bytes (Fieldata characters) to the end of the buffer to be converted.

outbuf

is a pointer to a variable that points to the first available byte in the output buffer.

outbytesleft

is an unsigned integer that indicates the number of available bytes (Fieldata characters) to the end of the buffer.

Return Values

This function always returns a value of zero (0).

Error Statuses

The string transliteration service routines can return the following error statuses in the error field:

- _E_ISL_2_SHORT
- _E_ISL_BAD_FROM
- _E_ISL_BAD_TO
- _E_ISL_NOT_IN_CCS
- _E_ISL_BAD_CD
- _E_ISL_CORRUPT_TABLE
- _E_ISL_CCS_NOT_FOUND
- _E_ISL_CCS_THE_SAME
- _E_ISL_CCS_BANK_FULL

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *open_ccs_to_ccs*(), *ccs_to_ccs*(), and *close_ccs_to_ccs*() to transliterate a character string from one CCS (ISO 646ES) to another (ISO 8859.1).

```
#include <iconv.h>
       .
       .
       .
int i;
iconv_t  cd;
    .
    .
    .
/* The ISO 646ES characters are in the buffer pointed to */
/* by inbuf_ptr, and the converted characters are placed */
/* in the buffer pointed to by outbuf_ptr.               */

cd = iconv_open("23","35");
if (cd == -1)
     printf("Conversion not available.\n");
else
```

```
{
i = iconv(cd,&inbuf_ptr,&inbuf_size,&outbuf_ptr,&outbuf_size);
if (i == -1)
   printf("Conversion failed.\n");
iconv_close(cd);
}
  .
  .
  .
```

3.9. getenv() Function

Name

getenv()

Description

getenv() retrieves the current value of the environment variable set at the run level. These variables are initially set by the default values from the user profile.

Declarations

```
#include <stdlib.h>
char *getenv(const char *name);
```

Input Values

<stdlib.h>

is the run-time library function call for general utilities. This is the standard header in which the service routine is declared.

name

is a pointer to a string containing the name of an environment variable.

Return Values

- If completed successfully, the service routine returns a pointer to a string containing the value for the specified environment variable name.
- If this name cannot be found, the service routine returns a null pointer.
- If a null pointer is returned, check the *errno* global variable to see if the value is other than zero (0). If so, an error has occurred.

Error Statuses

getenv() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_BADCHAR
- _E_ISL_VAL_BADCHAR

- _E_ISL_NAME_NOT_FOUND
- _E_ISL_GET$ENV_ERROR

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *getenv*() to determine if the LC_COLLATE environment variable is set to locale fr_FR.8859-1 at the run level. (This locale specifies French as spoken in France.)

```
char *p;

/* Is the French collation being used? */
p = getenv("LC_COLLATE");
if (p != NULL &&
   strcmp(p,"fr_FR.8859-1") = 0)
   printf("oui\n");
else
   printf("non\n");
```

3.10. getenvsys() Function

Name

getenvsys()

Description

getenvsys() retrieves the current value of the environment variable set at the system level.

Declarations

```
#include <stdlib.h>
char *getenvsys(const char *name);
```

Input Values

<stdlib.h>

is the run-time library function call for general utilities. This is the standard header in which the service routine is declared.

name

is a pointer to a string containing the name of an environment variable.

Return Values

- If completed successfully, the service routine returns a pointer to a string containing the value for the specified environment variable name.
- If this name cannot be found, the service routine returns a null pointer.

- If a null pointer is returned, check the *errno* global variable to see if the value is other than zero (0). If so, an error has occurred.

Error Statuses

getenvsys() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_BADCHAR
- _E_ISL_VAL_BADCHAR
- _E_ISL_NAME_NOT_FOUND
- _E_ISL_GET$ENV_ERROR

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *getenvsys*() to determine if the LC_COLLATE environment variable is set to the locale fr_FR.8859-1 at the system level. (This locale specifies French as spoken in France.)

```
char *p;

/* Is the French collation being used? */
p = getenvsys("LC_COLLATE");
if (p != NULL &&
   strcmp(p,"fr_FR.8859-1") == 0)
   printf("oui\n");
else
   printf("non\n");
```

3.11. isalnum() Function

Name

isalnum()

Description

isalnum() tests for any character in the alphanumeric character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the alpha and digit categories in the locale definition file.

Declarations

```
#include <ctype.h>
int isalnum(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

The code example shows how to use *isalnum*() to determine if an input character is alphanumeric.

```
#include <ctype.h>
     .
     .
     .
int c;
     .
     .
     .

if (isalnum(c))
       printf("The character is alphanumeric \n");
     .
     .
     .
```

3.12. isalpha() Function

Name

isalpha()

Description

isalpha() tests for any character in the alphabetic character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the upper, lower, and letter categories in the locale definition file.

Declarations

```
#include <ctype.h>
int isalpha(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *iisalpha*() to determine if an input character is alphabetic.

```
#include <ctype.h>
    .
    .
    .
int c;
    .
    .
    .
if (isalpha(c))
      printf("The character is alphabetic \n");
    .
```

.
.

3.13. iscntrl() Function

Name

iscntrl()

Description

iscntrl() tests for any character in the control character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *cntrl* category in the locale definition file.

Declarations

```
#include <ctype.h>
int iscntrl(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *iscntrl*() to determine if an input character is a control character.

```
#include <ctype,h>
        .
        .
        .
int c;
   .
   .
   .
if (iscntrl(c))
      printf("The character is a control character \n");
                          .
                          .
                          .
```

3.14. isdigit() Function

Name

isdigit()

Description

isdigit() tests for any character as a decimal digit in the numeric character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the digit category in the locale definition file.

Declarations

```
#include <ctype.h>
int isdigit(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.

- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *isdigit*() to determine if an input character is a decimal digit (numeric).

```
#include <ctype,h>
     .
     .
     .
int c;
     .
     .
     .
if (isdigit(c))
      printf("The character is a digit \n");
                  .
                  .
                  .
```

3.15. isgraph() Function

Name

isgraph()

Description

isgraph() tests for any character in the graphic character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the graph category in the locale definition file.

Declarations

```
#include <ctype.h>
int isgraph(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *isgraph*() to determine if an input character is a graphic character.

```
#include <ctype,h>
     .
     .
     .
int c;
    .
    .
    .
if (isgraph(c))
      printf("The character is graphical \n");
                .
                .
                .
```

3.16. islower() Function

Name

islower()

Description

islower() tests for any character in the lowercase character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the lower category in the locale definition file.

Declarations

```
#include <ctype.h>
int islower(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code routine shows how to use *islower*() to determine if an input character is lowercase and alphabetic.

```
#include <ctype,h>
     .
     .
     .
int c;
     .
     .
     .
if (islower(c))
      printf("The character is lowercase \n");
                  .
                  .
                  .
```

3.17. isprint() Function

Name

isprint()

Description

isprint() tests for any character in the printable character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the print category in the locale definition file.

Declarations

```
#include <ctype.h>
int isprint(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *isprint*() to determine if an input character is printable.

```
#include <ctype,h>
     .
     .
     .
int c;
```

```
      .
      .
      .
if (isprint(c))
      printf("The character is printable \n");
                  .
                  .
                  .
```

3.18. ispunct() Function

Name

ispunct()

Description

ispunct() tests for any character in the punctuation character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the punct category in the locale definition file.

Declarations

```
#include <ctype.h>
int ispunct(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *ispunct*() to determine if an input character is a punctuation character.

```
#include <ctype,h>
     .
     .
     .
int c;
     .
     .
     .
if (ispunct(c))
      printf("The character is punctuation \n");
                  .
                  .
                  .
```

3.19. isspace() Function

Name

isspace()

Description

isspace() tests for any character in the white space character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the space category in the locale definition file.

Declarations

```
#include <ctype.h>
int isspace(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.

- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *isspace*() to determine if an input character is a white space character.

```
#include <ctype,h>
     .
     .
     .
int c;
     .
     .
     .
if (isspace(c))
      printf("The character is a space character \n");
                  .
                  .
                  .
```

3.20. isupper() Function

Name

isupper()

Description

isupper() tests for any character in the uppercase character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the upper category in the locale definition file.

Declarations

```
#include <ctype.h>
int isupper(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *isupper*() to determine if an input character is uppercase and alphabetic.

```
#include <ctype,h>
      .
      .
      .
int c;
      .
      .
      .
if (isupper(c))
       printf("The character is uppercase \n");
                    .
                    .
                    .
```

3.21. isxdigit() Function

Name

isxdigit()

Description

isxdigit() tests for any character in the hexadecimal digit (xdigit) character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the xdigit category in the locale definition file.

Declarations

```
#include <ctype.h>
int isxdigit(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which the specific character class function is declared.

c

is the character to test.

Return Values

- If the character passed is the correct character class, a non-zero value is returned.
- If the character passed is not the correct character class, a zero (0) value is returned.
- If an error occurs in the routine, the *errno* global variable is set to indicate the error.

Error Status

There is one error status (_E_ISL_CORRUPT_TABLE) set in the *errno* global variable if the character class service routines fail: See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *isxdigit*() to determine if an input character is a hexadecimal digit.

```
#include <ctype,h>
     .
     .
     .
int c;
     .
     .
     .
if (isxdigit(c))
      printf("The character is a hexadecimal character \n");
                  .
                  .
                  .
```

3.22. locale_name_to_ccs_num() Function

Name

lnm2cnb()

Description

locale_name_to_ccs_num() returns the coded character set (CCS) number associated with the locale name.

Declarations

```
#include <locale.h>
int lnm2cnb(char *locnam, int ccsformat);
```

Input Values

<locale.h>

is the run-time library function call for locale information. This is the standard header in which the locale name is declared.

locnam

is a pointer to the string containing the locale name. The locale name is case-sensitive and must correspond to a LOCALE_NAME entity in the Repository and a locale that is active on the system.

ccsformat

is the kind of CCS number provided. It can be in the FMTCCS, FMTISO, or FMTSDF formats.

FMTCCS is the Unisys CCS number for use with *ccs_to_ccs*() transliteration.

FMTISO is the ISO CCS number for interoperability use.

FMTSDF is the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *locale_num_to_css_num*().

- If completed successfully, this function returns the locale name corresponding to the specified locale number.
- If an error occurs, the function sets the *errno* global variable to non-zero, and the value for *locnam* is undefined.

Error Statuses

locale_name_to_ccs_num() can return the following error statuses in the *errno* global variable:

- _E_ISL_LOCALE_NOT_FOUND
- _E_ISL_CCS_FORM_BAD

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *locale_name_to_ccs_num*() to return the CCS number associated with the locale name.

```
#include <locale.h>
     .
     .
     .
int ccsnum;
    .
    .
    .
ccsnum = lnm2cnb("fr_FR.8859-1", FMTSDF);
                .
                .
                .
```

3.23. locale_name_to_locale_num() Function

Name

lnm2lnb()

Description

locale_name_to_locale_num() transforms a locale name to its corresponding locale number.

Declarations

```
#include <locale.h>
int lnm2lnb(char *locnam);
```

Input Values

<locale.h>

is the run-time library function call for locale information. This is the standard header in which the locale name is declared.

locnam

is a pointer to the string containing the locale name. This name is required by the *setlocale*() service routine. It is case-sensitive and must correspond to a LOCALE_NAME entity in the Repository and a locale that is active on the system.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *locale_name_to_locale_num*().

- If completed successfully, this function returns the locale number, a numeric code corresponding to the specified locale name. This number can be stored in an MSAM file or in a Network Database Server data base record.
- If an error occurs, the function sets *errno* and returns a -1.

Error Statuses

locale_name_to_locale_num() can return the _E_ISL_LOCALE_NOT_FOUND error status in the *errno* global variable. See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *locale_name_to_locale_num*() to transform a locale name to its corresponding locale number.

```
#include <locale.h>
        .
        .
        .
int localenum;
        .
        .
        .
localenum = lnm2lnb("fr_FR.8859-1");
        .
        .
        .
```

3.24. locale_num_to_ccs_num() Function

Name

lnb2cnb()

Description

locale_num_to_ccs_num() returns the coded character set (CCS) number associated with the locale number.

Declarations

```
#include <locale.h>
int lnb2cnb(int locnum, int ccsformat);
```

Input Values

<locale.h>

is the run-time library function call for locale information. This is the standard header in which the locale number is declared.

locnum

is the number of a locale that is active on the system.

ccsformat

is the kind of CCS number provided. It can be in the FMTCCS, FMTISO, or FMTSDF formats.

FMTCCS is the Unisys CCS number for use with *ccs_to_ccs*() transliteration.

FMTISO is the ISO CCS number for interoperability use.

FMTSDF is the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *locale_name_to_ccs_num*().

- If completed successfully, this function returns the CCS number, the numeric code of the CCS associated with the specified locale.
- If an error occurs, the function sets *errno* and returns a -1.

Error Statuses

locale_num_to_ccs_num() can return the following error statuses in the *errno* global variable:

- _E_ISL_LOCALE_NOT_FOUND
- _E_ISL_NUM_FORM_BAD

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *locale_num_to_ccs_num*() to return the CCS number associated with the locale number.

```
#include <locale.h>
      .
      .
      .
int ccsnum;
      .
      .
      .
ccsnum = lnb2cnb(4, FMTSDF);
      .
      .
      .
```

3.25. locale_num_to_locale_name() Function

Name

lnb2lnm()

Description

locale_num_to_locale_name() transforms a locale number to its corresponding locale name.

Declarations

```
#include <locale.h>
void lnb2lnm(int locnum, char *locnam, size_t n);
```

Input Values

<locale.h>

is the run-time library function call for locale information. This is the standard header in which the locale number is declared.

locnum

is a locale number that corresponds to a specific locale on the system.

locnam

is a pointer to the resulting locale name. This name is required by the *setlocale*() service routine. It is case-sensitive and must correspond to a LOCALE_NAME entity in the Repository and a locale that is active on the system.

n

is the number of bytes that can be placed in the resulting locale name, including the terminating null byte.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *locale_num_to_locale_name*().

- If completed successfully, this function returns the locale name corresponding to the specified locale number.
- If an error occurs, the function sets the *errno* global variable to non-zero, and the value for *locnam* is undefined.

Error Statuses

locale_num_to_locale_name() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_LOCALE_NOT_FOUND

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *locale_num_to_locale_name*() is used to transform a locale number to its corresponding locale name.

```
#include <locale.h>
      .
      .
      .
char *lnm;
      .
      .
      .
lnb2lnm(6, lnm, 17);
      .
      .
      .
```

3.26. name_and_number Packet Interface

Name

isl_nmnb()

Description

The *name_and_number* packet interface provides a general interface for obtaining name-and-number information for various locales and CCSs. The Virtual Machine for the Java™ Platform on ClearPath OS 2200 can use this interface to obtain this information, which also includes the number of bits in a character for the requested locale or CCS.

Note: *All locales have an associated CCS number. However, not all CCSs are associated with a locale.*

Declarations

```
#include <locale.h>
void * isl_nmnb(&isl_result_status, &isl_nmnb_information_packet);
```

Error Statuses

The *name_and_number* packet interface can return the following error status in the *errno* global variable:

- _E_ISL_INVALID_PACKET_VERSION
- _E_ISL_TOO_MANY_PARAMETERS
- _E_ISL_TOO_FEW_PARAMETERS
- _E_ISL_LOCALE_NOT_FOUND
- _E_ISL_CCS_NOT_FOUND
- _E_ISL_CCS_NOT_UNIQUE
- _E_ISL_CCS_FORM_BAD
- _E_ISL_INVALID_NMNB_ID
- _E_ISL_INVALID_NMNB_NAME_LENGTH

See 3.2 for additional information about the error statuses.

Information Packet Structure

See the NAME_AND_NUMBER Packet Interface in the COBOL section for the list of fields in the NAME_AND_NUMBER information packet.

The following code defines the structure of the *name_and_number* information packet for internationalized OS 2200 applications that are written in C:

```
typedef struct
 {  unsigned isl_nmnb_reserved             : 18;
    unsigned isl_nmnb_identifier           : 6;
    unsigned isl_nmnb_char_bit_size        : 6;
    unsigned isl_nmnb_version              : 6;
    unsigned isl_nmnb_number;
    unsigned isl_nmnb_ccs_name_length;
    unsigned isl_nmnb_ccs_number;
    unsigned isl_nmnb_ccs_iso_number;
    unsigned isl_nmnb_ccs_sdf_number;
    char     isl_nmnb_ccs_name[16];
    unsigned isl_nmnb_locale_name_length;
    unsigned isl_nmnb_locale_number;
    char     isl_nmnb_locale_name[16];
  } isl_nmnb_information_packet;
#define isl_nmnb_information_packet_length 16

/* ISL_NMNB_IDENTIFIER field values */
```

```
#define isl_ccs_nb_id 1
#define isl_ccs_iso_nb_id 2
#define isl_ccs_sdf_nb_id 3
#define isl_locale_nb_id 4
#define isl_ccs_nm_id 5
#define isl_locale_nm_id 6
```

Sample Code

This code example shows how to use the *name_and_number* packet interface to obtain CCS and locale name-and-number information.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <locale.h>

main()
{
isl_nmnb_information_packet nmnb_pkt;
isl_result_status stat;
printf("<start of CCISLNMNB01>\n");

/* CCS number input */
printf("*** test of CCS number input ***\n");
nmnb_pkt.isl_nmnb_reserved = 0;
nmnb_pkt.isl_nmnb_version = 1;
nmnb_pkt.isl_nmnb_identifier = isl_ccs_nb_id;
nmnb_pkt.isl_nmnb_number = 35;
isl_nmnb(&stat, &nmnb_pkt);
if (errno == 0) {
  printf("char bit size = %d\n",nmnb_pkt.isl_nmnb_char_bit_size);
  printf("ccs iso number = %d\n",nmnb_pkt.isl_nmnb_ccs_iso_number);
  printf("ccs sdf number = %d\n",nmnb_pkt.isl_nmnb_ccs_sdf_number);
  printf("ccs name length = %d\n",nmnb_pkt.isl_nmnb_ccs_name_length);
  printf("ccs name %s\n",nmnb_pkt.isl_nmnb_ccs_name);
  }
else  {
  printf("isl_nmnb returned unexpected results status =
%d\n",stat.message_id);
  }

/* CCS ISO number input */
printf("*** test of CCS ISO number input ***\n");
nmnb_pkt.isl_nmnb_reserved = 0;
nmnb_pkt.isl_nmnb_version = 1;
nmnb_pkt.isl_nmnb_identifier = isl_ccs_iso_nb_id;
nmnb_pkt.isl_nmnb_number = 210;
isl_nmnb(&stat, &nmnb_pkt);
if (errno == 0) {
  printf("char bit size = %d\n",nmnb_pkt.isl_nmnb_char_bit_size);
```

```
  printf("ccs number = %d\n",nmnb_pkt.isl_nmnb_ccs_number);
  printf("ccs sdf number = %d\n",nmnb_pkt.isl_nmnb_ccs_sdf_number);
  printf("ccs name length = %d\n",nmnb_pkt.isl_nmnb_ccs_name_length);
  printf("ccs name %s\n",nmnb_pkt.isl_nmnb_ccs_name);
  }
else  {
  printf("isl_nmnb returned unexpected results status =
%d\n",stat.message_id);
  }

/* CCS SDF number input */
printf("*** test of CCS SDF number input ***\n");
nmnb_pkt.isl_nmnb_reserved = 0;
nmnb_pkt.isl_nmnb_version = 1;
nmnb_pkt.isl_nmnb_identifier = isl_ccs_sdf_nb_id;
nmnb_pkt.isl_nmnb_number = 61;
isl_nmnb(&stat, &nmnb_pkt);
if (errno == 0) {
  printf("char bit size = %d\n",nmnb_pkt.isl_nmnb_char_bit_size);
  printf("ccs iso number = %d\n",nmnb_pkt.isl_nmnb_ccs_iso_number);
  printf("ccs number = %d\n",nmnb_pkt.isl_nmnb_ccs_number);
  printf("ccs name length = %d\n",nmnb_pkt.isl_nmnb_ccs_name_length);
  printf("ccs name %s\n",nmnb_pkt.isl_nmnb_ccs_name);
  }
else  {
  printf("isl_nmnb returned unexpected results status =
%d\n",stat.message_id);
  }

/* Locale number input */
printf("*** test of Locale number input ***\n");
nmnb_pkt.isl_nmnb_reserved = 0;
nmnb_pkt.isl_nmnb_version = 1;
nmnb_pkt.isl_nmnb_identifier = isl_locale_nb_id;
nmnb_pkt.isl_nmnb_number = 35;
isl_nmnb(&stat, &nmnb_pkt);
if (errno == 0) {
  printf("char bit size = %d\n",nmnb_pkt.isl_nmnb_char_bit_size);
  printf("ccs number = %d\n",nmnb_pkt.isl_nmnb_ccs_number);
  printf("ccs iso number = %d\n",nmnb_pkt.isl_nmnb_ccs_iso_number);
  printf("ccs sdf number = %d\n",nmnb_pkt.isl_nmnb_ccs_sdf_number);
  printf("ccs name length = %d\n",nmnb_pkt.isl_nmnb_ccs_name_length);
  printf("ccs name %s\n",nmnb_pkt.isl_nmnb_ccs_name);
  printf("locale number = %d\n",nmnb_pkt.isl_nmnb_locale_number);
  printf("locale name length =
%d\n",nmnb_pkt.isl_nmnb_locale_name_length);
  printf("locale name %s\n",nmnb_pkt.isl_nmnb_locale_name);
  }
else  {
  printf("isl_nmnb returned unexpected results status =
%d\n",stat.message_id);
  }
```

```
/* CCS name input */
printf("*** test of CCS name input *** \n");
nmnb_pkt.isl_nmnb_reserved = 0;
nmnb_pkt.isl_nmnb_version = 1;
nmnb_pkt.isl_nmnb_identifier = isl_ccs_nm_id;
nmnb_pkt.isl_nmnb_ccs_name_length = 6;
strcpy(nmnb_pkt.isl_nmnb_ccs_name,"LETS-J");
isl_nmnb(&stat, &nmnb_pkt);
if (errno == 0) {
  printf("char bit size = %d\n",nmnb_pkt.isl_nmnb_char_bit_size);
  printf("ccs number = %d\n",nmnb_pkt.isl_nmnb_ccs_number);
  printf("ccs iso number = %d\n",nmnb_pkt.isl_nmnb_ccs_iso_number);
  printf("ccs sdf number = %d\n",nmnb_pkt.isl_nmnb_ccs_sdf_number);
  }
else  {
  printf("isl_nmnb returned unexpected results status =
%d\n",stat.message_id);
  }

/* Locale name input */
printf("*** test of Locale name input *** \n");
nmnb_pkt.isl_nmnb_reserved = 0;
nmnb_pkt.isl_nmnb_version = 1;
nmnb_pkt.isl_nmnb_identifier = isl_locale_nm_id;
nmnb_pkt.isl_nmnb_locale_name_length = 11;
strcpy(nmnb_pkt.isl_nmnb_locale_name,"en_US.ASCII");
isl_nmnb(&stat, &nmnb_pkt);
if (errno == 0) {
  printf("char bit size = %d\n",nmnb_pkt.isl_nmnb_char_bit_size);
  printf("ccs number = %d\n",nmnb_pkt.isl_nmnb_ccs_number);
  printf("ccs iso number = %d\n",nmnb_pkt.isl_nmnb_ccs_iso_number);
  printf("ccs sdf number = %d\n",nmnb_pkt.isl_nmnb_ccs_sdf_number);
  printf("ccs name length = %d\n",nmnb_pkt.isl_nmnb_ccs_name_length);
  printf("ccs name %s\n",nmnb_pkt.isl_nmnb_ccs_name);
  printf("locale number = %d\n",nmnb_pkt.isl_nmnb_locale_number);
  }
else  {
  printf("isl_nmnb returned unexpected results status =
%d\n",stat.message_id);
  }

printf("<end of CCISLNMNB01>\n");
return(0);
}
```

3.27. nl_langinfo() Function

Name

nl_langinfo()

Description

nl_langinfo() returns a string containing relevant information about the particular language or cultural area defined in the program's locale.

Declarations

```
#include <langinfo.h>
char *nl_langinfo(nl_item item);
```

Input/Output Values

<langinfo.h>

is the run-time library function call for language information. This is the standard header in which the definitions for locale data items are specified.

item

is a constant indicating which information defined in the locale to return.

Return Values

In a locale where the particular locale data is not defined, *nl_langinfo*() returns a pointer to the corresponding string in the "C" locale.

In all locales, *nl_langinfo*() returns a pointer to an empty string if *item* contains an invalid setting.

Error Statuses

nl_langinfo() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_CORRUPT_TABLE
- _E_ISL_ASTORE_READ
- _E_ISL_ASTORE_WRITE
- _E_ISL_CREATE_BANK
- _E_ISL_CKRS_ERROR
- _E_ISL_BAD_ITEM

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *nl_langinfo*() to return a string containing information about a particular language or cultural area defined in the program's locale.

This example illustrates a call to *nl_langinfo*() to return the abbreviated name of the second day of the week (lundi), based on the locale fr_FR.8859-1, which specifies French as spoken in France.

If *nl_langinfo*() is not successful, perror() can be called to display an error message.

```
#include <stdio.h>
#include <errno.h>
#include <langinfo.h>

int main()
{
   char *s;

   errno = 0;
   s = nl_langinfo(ABDAY_2);
   if (errno = 0)
      printf("%s", s);

   return(0);
}
```

Assuming that the locale for LC_TIME is "fr_FR.8859-1", the following output is displayed:

```
Lundi
```

3.28. open_ccs_to_ccs() Function

Names

iconv_open()

Description

open_ccs_to_ccs() is one of the three service routines used to transliterate a character string from one coded character set (CCS).

Declarations

```
#include <iconv.h>
iconv_t iconv_open(const char *tocode, const char *fromcode);

size_t iconv(iconv_t cd, const char **inbuf, size_t *inbytesleft,
char **outbuf, size_t *outbytesleft);

int iconv_close(iconv_t cd);
```

Input Values

<iconv.h>

is the run-time library function call for character conversion. This is the standard header in which the three string transliteration functions are declared.

tocode

is a pointer to a string that identifies the CCS resulting from the transliteration. This is a decimal number.

fromcode

is a pointer to a string that identifies the CCS to be transliterated. This is a decimal number.

cd

is the conversion descriptor (internal identifier for the transliteration tables) returned by *open_ccs_to_ccs*().

inbuf

is a pointer to a variable that points to the first character in the input buffer.

inbytesleft

is an unsigned integer that indicates the number of bytes (Fieldata characters) to the end of the buffer to be converted.

outbuf

is a pointer to a variable that points to the first available byte in the output buffer.

outbytesleft

is an unsigned integer that indicates the number of available bytes (Fieldata characters) to the end of the buffer.

Return Values

- If completed successfully, this function returns a conversion descriptor for use on subsequent calls to *ccs_to_ccs*(). This is an internal identifier for the transliteration tables.
- If not completed successfully, this function returns (iconv_t)-1 and sets the *errno* global variable to indicate the error.

Error Statuses

The string transliteration service routines can return the following error statuses in the error field:

- _E_ISL_2_SHORT
- _E_ISL_BAD_FROM
- _E_ISL_BAD_TO
- _E_ISL_NOT_IN_CCS
- _E_ISL_BAD_CD

- _E_ISL_CORRUPT_TABLE
- _E_ISL_CCS_NOT_FOUND
- _E_ISL_CCS_THE_SAME
- _E_ISL_CCS_BANK_FULL

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *open_ccs_to_ccs*(), *ccs_to_ccs*(), and *close_ccs_to_ccs*() to transliterate a character string from one CCS (ISO 646ES) to another (ISO 8859.1).

```
#include <iconv.h>
       .
       .
       .
int i;
iconv_t  cd;
    .
    .
    .
/* The ISO 646ES characters are in the buffer pointed to */
/* by inbuf_ptr, and the converted characters are placed */
/* in the buffer pointed to by outbuf_ptr.               */

cd = iconv_open("23","35");
if (cd == -1)
     printf("Conversion not available.\n");
else
  {
  i = iconv(cd,&inbuf_ptr,&inbuf_size,&outbuf_ptr,&outbuf_size);
  if (i == -1)
     printf("Conversion failed.\n");
  iconv_close(cd);
  }
    .
    .
    .
```

3.29. putenv() Function

Name

putenv()

Description

putenv() sets the current value of the environment variable for the run level.

Declarations

```
#include <stdlib.h>
int putenv(const char *string);
```

Input Values

putenv: Input Values (C Compiler)

<stdlib.h>

is the run-time library function call for general utilities. This is the standard header in which *putenv*() is declared.

string

is a pointer to a string of the form *name=value*. The name specifies the environment variable, and the value specifies the value of this variable.

Return Values

- If completed successfully, *putenv*() returns zero (0).
- If not completed successfully, *putenv*() returns a non-zero value and sets the *errno* global variable to non-zero to indicate the error.

Error Statuses

putenv() can return the following error statuses in the *errno* global variable:

- _E_ISL_BADCHAR
- _E_ISL_VAL_BADCHAR
- _E_ISL_ENV_AREA_FULL
- _E_ISL_BAD_TIP_EV
- _E_ISL_VAL_2_LONG
- _E_ISL_PUT$ENV_ERROR

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *putenv*() to set the LC_COLLATE environment variable value to the locale fr_FR.8859-1. (This locale specifies French as spoken in France.)

```
if (putenv("LC_COLLATE=fr_FR.8859-1") != 0)

/* If PUTENV is not successful, the perror() function is */
/* called to display an error message. */

   perror(NULL);
```

3.30. setlocale() Function

Name

setlocale() – set and query category values.

Description

The *setlocale*() function is used to set category values and query locale information for a user program.

Once called by the program, *setlocale*() establishes the locale for the current activity. This operation includes such functions as specifying the locale name, and the length and storage location for this name.

Setting Category Values

The setting of category values is indeterminate when a user program begins execution. This program calls *setlocale*() to set all or a portion of these values, as specified by the *category* and *locale* arguments.

At program startup, the equivalent of `setlocale(LC_ALL, "C");` is executed. This means that all categories are initialized to the "C" locale when the program begins.

Querying for Locale Information

setlocale() can query information about all or a portion of the program's locale.

- When the locale argument is a NULL pointer, a query of the current locale is requested.
- The string "_QUERY" may also be used to query the current locale. This is a Unisys extension.
- A query of the LC_ALL category returns the locale name for the LC_COLLATE category.

Declarations

```
#include <locale.h>
char *setlocale(int category, const char *locale);
```

Input Values

<locale.h>

is the run-time library function call for locale information. This is the standard header in which the macro definitions for the categories are specified.

category

is an integer that specifies the portion of the program's locale to be set or queried. There are seven allowable categories: 0=LC_ALL, 1=LC_COLLATE, 2=LC_CTYPE, 3=LC_MESSAGES, 4=LC_MONETARY, 5=LC_NUMERIC, and 6=LC_TIME.

locale

is a pointer to a string that specifies the name of a locale. The values are either "_DEFAULT" or "_QUERY". The maximum length for a locale name is 16 characters. When dealing with locale names, *setlocale*() is case-sensitive and ignores trailing spaces.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *setlocale*().

- If completed successfully, *setlocale*() returns a pointer to a string containing the name of the locale that was set or queried for by *locale*. The value of the string is such that a subsequent call with that string value and its associated *category* will restore that part of the program's locale.
- If the locale name cannot be found, *setlocale*() returns a null pointer. A null pointer (or "_QUERY") as the *locale* argument causes *setlocale*() to return a pointer to a string containing the name of the locale associated with the specified *category*. The locale is not changed.
- If a null pointer is returned, check *errno*. If the value is other than zero, an error has occurred.

Error Statuses

The *setlocale*() function can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_BADCHAR
- _E_ISL_VAL_BADCHAR
- _E_ISL_LOCALE_NOT_FOUND
- _E_ISL_BAD_CATEGORY
- _E_ISL_CORRUPT_TABLE
- _E_ISL_GET$ENV_ERROR

See 3.2 for additional information about the error statuses.

Example 1: Querying Current Locale for LC_COLLATE (C Compiler)

The following code can be used to call *setlocale*() to query the program's current locale for the category LC_COLLATE:

```
p = setlocale(LC_COLLATE, NULL);
```

where

setlocale()

returns a pointer to a string corresponding to the program's current locale.

LC_COLLATE

is the category for the current locale. This string value may be used by a subsequent call to *setlocale*() to reset the locale for this category.

NULL

is the return value from *setlocale*(). This is a pointer to a static area within the function, but it is not guaranteed to remain unchanged. It may be modified by a subsequent call to *setlocale*().

Note: *If your purpose in calling setlocale*() *is to save the value of the program's current I18N environment so that it can be changed and reset later, copy the return value to an array of char in the calling program.*

Example 2: Setting the I18N Environment (C Compiler)

There are three ways to set the program's international environment with *setlocale*().

Method 1: Use setlocale(*category*, *string*)

A specific *category* in the program's I18N environment will be set to a specific value corresponding to the value of *string*:

```
setlocale(LC_ALL, "fr_FR.8859-1").
```

This means that all categories will be set to the locale that corresponds to the string "fr_FR.8859-1". This string corresponds to the French language as spoken in France using the ISO 8859.1 code set.

Method 2: Use setlocale(*category*, "C")

A specific *category* in the program's I18N environment will be set to the "C" locale. This locale corresponds to the minimal environment needed to support the C programming language.

Method 3: Use setlocale(*category*, "")

A specific *category* in the program's I18N environment will be set to the value of the environment variables.

Expanded Example 2: Setting the I18N Environment (C Compiler)

1. Assume that the environment variables are set as follows:

```
LC_ALL=\n
LC_COLLATE=de_DE.8859-1
LC_CTYPE=\n
LANG=fr_FR.8859-1
```

A setlocale(LC_ALL,""); call will cause the following to occur:

a) The program's LC_COLLATE category to be set to the de_DE.8859-1 locale, which corresponds to the German language as spoken in Germany; and

b) The LC_CTYPE category to be set to the fr_FR.8859-1 locale, which corresponds to the French language as spoken in France.

Both languages use the ISO 8859.1 code set.

2. Now assume that the environment variables have the same values except

```
LC_ALL=no_NO.8859-1
```

A `setlocale(LC_ALL,"");` call will cause both of the program's categories (LC_COLLATE and LC_CTYPE) to be set to the no_NO.8859-1 locale, which corresponds to the Norwegian language as spoken in Norway.

3. Now see how a program can initialize a program's international environment for one language, while selectively modifying the program's locale so that the collation operations can be applied to strings recorded in a different language.

```
setlocale(LC_ALL, "de_DE.8859-1");
setlocale(LC_COLLATE, "fr_FR.8859-1");
```

3.31. strfmon() Function

Name

strfmon()

Description

strfmon() is used to convert numeric monetary values to a string representation based on a specified format and the locale information defined by the LC_MONETARY category.

Declarations

```
#include <monetary.h>
char *ssize_t strfmon(char *s, size_t maxsize, const char
                      *format,...);
```

Input Values

<monetary.h>

is the run-time library function call for monetary information. This is the standard header in which the *strfmon*() procedure is specified.

s

is a pointer to where the resulting formatted monetary string will be placed.

maxsize

is the maximum number of bytes that can be placed in s (including the terminating NULL byte).

format

is the format string; this string contains plain characters and conversion specifications.

Plain Characters: Plain characters are copied unchanged to s. They include the terminating NULL byte used by the C Compiler.

Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

...

is zero or more arguments that are converted and formatted. Each argument must be declared as double. For details, go to the Conversion Specification Table.

Conversion Specification Table

Table 3–2 shows the sequence of arguments in a conversion specification. Each conversion specification results in the fetching of zero or more sequential arguments that are converted and formatted.

- Each argument must be a double-precision floating point number.
- If excess arguments remain after the format string is exhausted, they are ignored.
- If the format string contains more conversion specifications than arguments, the results are unpredictable. With the C compiler, an error is returned.

Note: *The % character and the conversion character are required; all others are optional.*

Table 3–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
1	% character	Required argument
2	flags	Optional arguments
	=f	A numeric fill character. This character (1) must be representable in a single byte; (2) does not affect field width filling, which always uses the space character; and (3) is ignored unless left precision is specified. The default is the space character.

Table 3–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
	^	Indicates that the currency amount should not be formatted with currency characters. If this flag is defined for the current locale, grouping characters are inserted.
	+ or (	Specifies the style for representing positive and negative currency amounts. Only one style can be specified. If +, the locale's equivalent of + and - are used. If (, negative amounts are enclosed in parentheses ( ). If neither style is specified, the default is +.
	!	Suppresses the currency symbol from output conversion.
	=	Specifies the alignment. If it is present, all fields are left justified.
3	field width (w)	Optional argument. A decimal digit that specifies a minimum field width in bytes. If the - flag is specified, the result is left-justified. If the - flag is not specified, the result is right-justified. The default is zero (0).
4	left precision (#n)	Optional argument. A pound sign followed by a decimal digit string that specifies the maximum number of digits expected to be formatted to the left of the radix character. It can be used to (1) keep formatted output from multiple calls to STRFMON aligned in the same columns and (2) fill unused positions with a special character. • If more than n digit positions are required, this value is ignored, and the excess positions are filled with numeric fill characters (=f). • If the grouping has not been suppressed with the ^ flag and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit. • To ensure alignment, any characters appearing before or after the number in the formatted output (such as currency or sign symbols) are padded as necessary with space characters to make their positive and negative formats an equal length.
5	right precision (.p)	Optional argument. A period followed by a decimal digit that specifies the number of digits after the radix character. If the value is zero, no radix character appears . The amount being formatted is rounded to the specified number of digits before formatting occurs. If right precision is not included, a default specified by the current locale is used.
6	conversion character	Required argument
	i	This argument should use the locale's international currency format. USA example: `USD 1,234.56`

Table 3–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
	n	This argument should use the locale's national currency format. USA example: `$1,234.56`
	%	This means no argument is converted. The entire conversion specification must be %%.

Return Values

If the total number of resulting bytes (including the terminating NULL byte) is not more than *maxsize*, the *strfmon*() function returns the number of bytes placed into the string pointed to by *s* (not including the terminating NULL byte).

Otherwise, a -1 is returned, the contents of the string are indeterminate, and *errno* is set to indicate the error.

Error Statuses

strfmon() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_BADCHAR
- _E_ISL_VAL_BADCHAR
- _E_ISL_LOCALE_NOT_FOUND
- _E_ISL_CORRUPT_TABLE
- _E_ISL_GET$ENV_ERROR
- _E_ISL_ASTORE_READ
- _E_ISL_ASTORE_WRITE
- _E_ISL_CREATE_BANK
- _E_ISL_CKRS_ERROR
- _E_ISL_CONV_SPEC
- _E_ISL_BAD_ARGS

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *strfmon*() to format a monetary value. If *strfmon*() is not successful, perror() can be called to display an error message.

```
#include <stdio.h>
#include <errno.h>
#include <monetary.h>
```

```
int main()
{
   char s[50];
   size_t maxsize = 50;
   char *format = {"Your total cost is %=*#5n\n"};
   double argument = -123.45;

   if ( strfmon(s, maxsize, format, argument) != -1 )
      printf("%s", s);

   return(0);
}
```

Assuming that the locale for LC_MONETARY is C, the following output is displayed:

```
Your total cost is -$***123.45
```

3.32. strftime() Function

Name

strftime()

Description

strftime() is used to convert date-and-time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

Declarations

```
#include <time.h>
size_t strftime(char *s, size_t maxsize, const char *format,
                const struct tm *timptr);
```

Input Values

<time.h>

is the run-time library function call for date and time information. This is the standard header in which the date/time structure is specified.

s

is a pointer to where the resulting formatted time string will be placed.

maxsize

is the maximum number of bytes that can be placed in s (including the terminating NULL byte).

format

is a format string containing plain characters and conversion specifications.

- Plain Characters: Plain characters are copied unchanged to s. They include the terminating NULL byte used by the C Compiler.
- Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

timptr

is a pointer to a time structure containing the values that will be used along with the program's locale to determine the appropriate conversions. To see a list of these values, go to the Date-and-Time Structures Table.

Conversion Specification Table

Each conversion specification (CS) is replaced by the appropriate characters listed in this table. These characters are determined by the current locale and by the values contained in the date-and-time structure. If a CS does not correspond to any of the ones listed in this table, the behavior is undefined.

Table 3–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%a	Weekday name, abbreviated	abday	tm_wday
%A	Weekday name, full	day	tm_wday
%b	Month name, abbreviated	abmon	tm_mon
%B	Month name, full	mon	tm_mon
%c	Appropriate date/time representation	d_t_fmt	Varies by locale
%C	Century number. To get this number, the year is divided by 100 and truncated to an integer as a decimal [00-99].	n/a	tm_year
%d	Day of month as decimal [01,31]	n/a	tm_mday
%D	Equivalent to %m/%d/%y	n/a	tm_mon, tm_mday, and tm_year.
%e	Day of month as decimal [1,31]. A single digit is preceded by a space.	n/a	tm_mday
%F	Equivalent to %Y-%m-%d (the ISO8601:2000 standard date format)	n/a	tm_year, tm_mon, and tm_mday
%g	Replaced by the last 2 digits of the week-based year as a decimal number [00-99].	n/a	tm_year, tm_yday, and tm_mday

Table 3–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%G	Replaced by the week-based year and as a decimal number.	n/a	tm_year, tm_yday, tm_mday
%h	Month name, abbreviated	abmon	tm_mon
%H	Hour on 24-hour clock as decimal [00,23]	n/a	tm_hour
%I	Hour on 12-hour clock as decimal [01,12]	n/a	tm_hour
%j	Day of year as decimal [001,366]	n/a	tm_yday
%m	Month as decimal [01,12]	n/a	tm_mon
%M	Minute as decimal [00,59]	n/a	tm_min
%n	New line character (012)	n/a	n/a
%p	Equivalent for ante-meridiem (AM) and post-meridiem (PM)	am_pm	tm_hour
%r	Time in AM and PM notation; equivalent of %I:%M:%S %p in the POSIX locale	t_fmt_ampm	Refer to individual CSs.
%R	Time in 24-hour notation (%H:%M)	n/a	tm_hour and tm_min.
%S	Second as decimal [00,60]	n/a	tm_sec
%t	Tab character	n/a	n/a
%T	Time (%H:%M:%S)	n/a	tm_hour, tm_min, and tm_sec.
%u	Weekday as decimal [1,7]	n/a	tm_wday
%U	Week number of the year as decimal [00,53] with Sunday as day 1	n/a	tm_yday and tm_wday
%V	Week number of the year as decimal [01,53] with Monday as day 1 If the week containing January 1 has four or more days in the new year, then it is considered week 1. If is does not, then it is the last week of the previous year, and the next week is week 1.	n/a	tm_year, tm_mday, and tm_yday
%w	weekday as decimal [0,6] with 0 representing Sunday	n/a	tm_wday

Table 3–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%W	Week number of the year as decimal [00,53] with Monday as day 1. All days in a new year preceding the first Sunday are considered to be in week 0.	n/a	tm_wday and tm_yday
%x	Appropriate date representation	d_fmt	Varies by locale
%X	Appropriate time representation	t_fmt	Varies by locale
%y	Year without century as decimal [00,99]	n/a	tm_year
%Y	Year with century as decimal [1998]	n/a	tm_year
%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm) or by no characters if no timezone is determinable.	n/a	tm_isdst
%Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.	n/a	tm_isdst
%%	%	n/a	n/a

Return Values

If the conversion is completed successfully, *strptime*() returns a pointer to the character that follows the last character parsed. Otherwise, a NULL pointer is returned.

Error Statuses

strftime() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_CORRUPT_TABLE
- _E_ISL_ASTORE_READ
- _E_ISL_ASTORE_WRITE
- _E_ISL_CREATE_BANK
- _E_ISL_CKRS_ERROR
- _E_ISL_CONV_SPEC
- _E_ISL_INVALID_DATE_TIME
- _E_ISL_SYSTEM$TIME_ERROR

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *strftime*() to return a formatted date and time, based on locale es_ES.8859-1 (Spanish as spoken in Spain). If *strftime*() is not successful, perror() can be called to display an error message.

```
#include <stdio.h>
#include <errno.h>
#include <time.h>

int main()
{
   char s[50];
   size_t maxsize = 50;
   char *format = {"Today is %X\n"};
   struct tm *timptr;
   time_t crnt_time;

   crnt_time = time(NULL);          /* get current calendar time */
   timptr = localtime(&crnt_time);  /* set local time in tm struct */
   if ( strftime(s, maxsize, format, timptr) != 0 )
      printf("%s", s);

   return(0);
}
```

Assuming that it is currently August 18, 1995 at 14:15:21, and the locale for LC_TIME is es_ES.8859-1, the following output is displayed:

```
Today is vie 18 ago 1995 14:15:21
```

%X for es_ES.8859-1 is represented as "%a %d %b %Y %T %Z".

3.33. string_collate() Function

Name

strcoll()

Description

string_collate() compares two character strings and returns their collating order. These strings can be up to 4096 characters long. The collating rules are determined by the current setting of the LC_COLLATE category.

Declarations

```
#include <string.h>
int strcoll(const char *s1, const char *s2);
```

Input Values

<string.h>

is the run-time library function call for string handling. This is the standard header in which *string_collate*() is declared.

s1

is a pointer to the first string to be compared.

s2

is a pointer to the second string to be compared.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *string_collate*().

- If completed successfully, *string_collate*() returns an integer greater than, equal to, or less than zero (0), according to whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*.
- If this integer cannot be found or an error occurs, *string_collate*() returns a null pointer and sets *errno*. The return value is undefined.
- If a null pointer is returned, check *errno* to see if the value is other than zero (0). If so, an error has occurred.

Error Statuses

string_collate() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_ENC_NOT_SUP
- _E_ISL_BADCHAR
- _E_ISL_VAL_BADCHAR
- _E_ISL_LOCALE_NOT_FOUND
- _E_ISL_CORRUPT_TABLE
- _E_ISL_STRING1_2_LONG
- _E_ISL_STRING2_2_LONG
- _E_ISL_GET$ENV_ERROR

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *string_collate*() to compare two character strings and return their collating order.

```
/* This routine determines if the string pointed to */
/* by "p" collates greater than the string pointed */
/* to by "q" for the current locale. */

if (strcoll(p, q) > 0)

   printf("p collates after q\n");
```

3.34. string_transform() Function

Name

strxfrm()

Description

string_transform() converts a character string into an ordering key. This string can be up to 4096 characters long. The transformation rules are determined by the current setting of the LC_COLLATE category.

Sizing the Output Buffer

The output buffer must be declared with sufficient size to hold the transformed string. The *string_transform*() service routine calculates the maximum size needed.

Process

1 The service routine passes zero (0) for both the *n* and the *s1* parameters.

2 The string pointed to by *s2* is transformed, and the resulting string is placed into the array pointed to by *s1*.

3 If the STRING COLLATE MAP (strcmp) function is applied to the transformed strings, it returns a value greater than, equal to, or less than zero (0). The same result would occur if *string_collate*() were applied to the original strings.

4 No more than *n* bytes are placed into the resulting array pointed to by *s1*, including the terminating null byte.

5 If copying takes place between objects that overlap, the behavior is undefined.

6 If *n* is 0, *s1* is permitted to be a null pointer. The I18N application can then determine the size of the *s1* array before making the transformation.

Example

The value of the following expression is the exact size of the array needed to hold the transformation of the string pointed to by *s2*.

```
1 + strxfrm(NULL, s2, 0)
```

Results

- If n = 0 and s1 = 0, *string_transform*() returns the maximum length for the converted string.

- If n <> 0 and s1 = 0, an error occurs in the calling sequence.
- If n = 0 and s1 <> 0, an error status (Buffer Too Short) is returned.

Declarations

```
#include <string.h>
int strxfrm(char *s1, const char *s2, size_t n);
```

Input Values

<string.h>

is the run-time library function call for string handling. This is the standard header in which *string_transform*() is declared.

s1

is a pointer to the resulting transformed string

s2

is a pointer to the string to be transformed.

n

is the number of bytes that can be placed in the resulting string pointed to by s1, including the terminating null byte.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *string_transform*().

- If completed successfully, *string_transform*() returns the length of the transformed string (not including the terminating null byte).
- If the value returned is *n* or more, the contents of the array pointed to by *s1* are indeterminate.
- If an error occurs, *string_transform*() sets *errno* to non-zero, and the return value is undefined.

Error Statuses

string_transform() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_ENC_NOT_SUP
- _E_ISL_BADCHAR
- _E_ISL_VAL_BADCHAR
- _E_ISL_LOCALE_NOT_FOUND
- _E_ISL_CORRUPT_TABLE

- _E_ISL_STRING1_2_LONG
- _E_ISL_GET$ENV_ERROR
- _E_ISL_ASTORE_READ
- _E_ISL_ASTORE_WRITE
- _E_ISL_CREATE_BANK
- _E_ISL_CKRS_ERROR

See 3.2 for additional information about the error statuses.

Sample Code

This code routine shows how to use *string_transform*() to convert a character string to an ordering key.

```
/* Call STRING_TRANSFORM twice. */
/* First, to transform the string pointed to by "p". */
/* Second, to transform the string pointed to by "q". */
/* The resulting transformed strings are pointed to */
/* by "pt" and "qt". */

strxfrm(pt, p, 100);
strxfrm(qt, q, 100);

/* Call the STRING_COLLATE MAP function to */
/* determine if the original string pointed to by "p" */
/* collates greater than the original string pointed */
/* to by "q" for the current locale.

if (strcmp(pt, qt) > 0)
   printf("p collates after q\n");
```

3.35. strptime() Function

Name

strptime()

Description

strptime() is used to convert date-and-time string characters to values based on a specified format and the locale information defined by the LC_TIME category.

Declarations

```
#include <time.h>
char *strptime(const char *buf, const char *format,
               const struct tm *timptr);
```

Input Values

<time.h>

is the run-time library function call for date and time information. This is the standard header in which the date/time structure is specified.

buf

is a pointer to the input string from which characters are converted to numeric values.

format

is the string describing the format of the input string; it is used along with the LC_TIME information to convert the string to integer values. The input string includes zero or more directives, which consist of one or more white-space characters and either a plain character or a conversion specification.

- Directives: The directives are the *strptime*() execution rules. For more details, go to the List of Directives in a Format String.
- Plain Character: A plain character is copied unchanged to the output buffer.
- Conversion Specification: A conversion specification consists of a % character and a terminating conversion character that determines the behavior of the specification. For details, see the *Conversion Specification Table* (Table 3–3).

timptr

is a pointer to a time structure into which the converted values are saved. To see a list of these values, see the *Date-and-Time Structures Table* (Table 3–1).

Return Values

If the conversion is completed successfully, *strptime*() returns a pointer to the character that follows the last character parsed. Otherwise, a NULL pointer is returned.

Error Statuses

strptime() can return the following error statuses in the *errno* global variable:

- _E_ISL_2_SHORT
- _E_ISL_BADCHAR
- _E_ISL_VAL_BADCHAR
- _E_ISL_LOCALE_NOT_FOUND
- _E_ISL_CORRUPT_TABLE
- _E_ISL_GET$ENV_ERROR
- _E_ISL_ASTORE_READ
- _E_ISL_ASTORE_WRITE
- _E_ISL_CREATE_BANK

- _E_ISL_CKRS_ERROR
- _E_ISL_CONV_SPEC
- _E_ISL_CHR_MISMATCH
- _E_ISL_INVALID_DATE_TIME

See 3.2 for additional information about the error statuses.

Sample Code

This code example shows how to use *strptime*() to convert a date/time string to numeric values, based on locale es_ES.8859-1 (Spanish as spoken in Spain). If *strftime*() is not successful, perror() can be called to display an error message.

```
#include <stdio.h>
#include <errno.h>
#include <time.h>

int main()
{
   char *buf = {"Today is vie 18 ago 1995 14:15:21 \n"};
   char *format = {"Today is %c \n"};
   struct tm *timptr;

   if (strptime(buf, format, timptr) == NULL)
      perror(NULL)

   return(0);
}
```

Assuming that the locale for LC_TIME is "es_ES.8859-1", the string pointed to by buf using the format string is converted to time values and saved in the structure pointed to by timptr.

```
timptr.tm_sec 21
timptr.tm_min 15
timptr.tm_hour  14
timptr.tm_mday  18
timptr.tm_mon 7
timptr.tm_year  95
timptr.tm_isdst -1
```

. %c for "es_ES.8859-1" is represented as "%a %d %b %Y %T %Z".

3.36. tolower() Function

Name

tolower()

Description

tolower() converts each character in a string to its corresponding lowercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

Declarations

```
#include <ctype.h>
int tolower(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which *tolower*() is declared.

c

is the character to be converted to lowercase.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *tolower*().

- If completed successfully, *tolower*() returns the lowercase character corresponding to the argument passed.
- If not completed successfully, *tolower*() returns the argument unchanged.
- If an error occurs, *errno* is set to non-zero to indicate the error.

Error Status

tolower() can return the _E_ISL_CORRUPT_TABLE error status in the *errno* global variable. See 3.2 for additional information about this error status.

Sample Code

The code example shows how to use *toupper*() to convert a input character to a lowercase alphabetic character. This example obtains the lowercase character a for the uppercase character A.

```
int c = 'A';
c = tolower (c);
```

3.37. toupper() Function

Name

toupper()

Description

toupper() converts each character in a string to its corresponding uppercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

Handling Characters with the int Argument

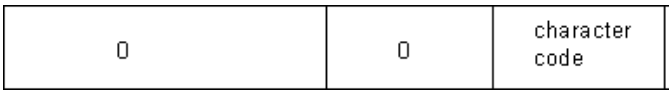
The *int* argument used to test for *toupper*() or *tolower*(). The value of *int* must be one of the following:

- Representable as an unsigned char.
- Equal to the value of the macro EOF. This value must be from -1 through 511 (octal 0777).

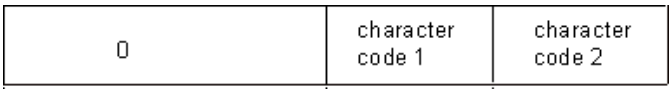
If the argument has any other value, the behavior is undefined. The character codes beyond 255 (octal 377) are undefined. No conversion will occur for these characters.

Handling Character Codes

When *toupper*() returns one character, the character code is returned as a quarter-word integer.



When the *toupper*() returns two characters, the character codes are returned as two quarter-word integers.



Declarations

```
#include <ctype.h>
int toupper(int c);
```

Input Values

<ctype.h>

is the run-time library function call for character classification. This is the standard header in which *toupper*() is declared.

c

is the character to be converted to uppercase.

Return Values

There is no return value reserved to indicate an error. For an application to check for error situations, it must first set the *errno* global variable to zero (0) and then call *toupper*().

- If completed successfully, *toupper*() returns the uppercase character corresponding to the argument passed.
- If not completed successfully, *toupper*() returns the argument unchanged.
- If an error occurs, *errno* is set to non-zero to indicate the error.

Error Status

toupper() can return the _E_ISL_CORRUPT_TABLE error status in the *errno* global variable. See 3.2 for additional information about this error status.

Sample Code

This code example shows how to use *toupper*() to convert a input character to an uppercase alphabetic character. The example obtains the uppercase character A for the lowercase character a.

```
int c = 'a';
c = toupper (c);
```

3.38. transform_length() Function

There is no C language binding to the *transform_length*() service routine. The *transform_length*() functions for the C Compiler are included in the *string_transform*() service routine.

Section 4
COBOL Compiler

This section describes the I18NLIB service routines available for the UCS COBOL language (COBOL Compiler). It also provides information about the COPY procedures that define the structures and the additional I18N language functionality added as a Unisys extension to perform I18N capabilities directly.

4.1. Compiler Features for Accessing I18NLIB Service Routines and Creating I18N Applications

Internationalizing COBOL Compiler Programs

You can internationalize these programs in one of two ways:

- By inserting a CALL statement to the desired I18NLIB service routine. For example, UCOB$SETLOC calls the service routine that establishes the current locale for an activity.
- By using the data types defined by the USAGE DISP-I18N and USAGE BYTE-I18N clauses.

Using the I18N-IO/NO-I18N-IO Options

There are two options that can be used to get code types and locale numbers: I18N-IO and NO-I18N-IO (the default).

- If the I18N-IO option is set, calls to the I18NLIB service routines are generated to get the CODE TYPE and LOCALE numbers of the locale in force.
- If the I18N-IO option is not set (NO-I18N-IO), calls to these service routines are not set, and CODE TYPE 1 (ASCII) and LOCALE 42 (en_US.ASCII) are always used. This setting is recommended for users who want only ASCII code types and locale numbers; it eliminates the overhead of I18NLIB calls.

4.2. I18N Features of COBOL Compiler Programs

The COBOL compiler provide the following features for creating I18N applications:

DISP-I18N USAGE clause

Defines a data type that allows the compiler to do comparisons in a culturally sensitive manner. The comparison rules in the current locale apply to all comparisons done with this data type.

BYTE-I18N USAGE clause

Defines a data item for containing the results of the UCOB$STRXFRM transformation.

SORT and MERGE statements

Enhance the SORT and MERGE interfaces to allow culturally sensitive sorting of records.

LOCALE clause on SELECT statement

Allows retrieval or setting of the locale associated with an ORGANIZATION INDEXED (MSAM) file.

CODE TYPE clause on SELECT statement

Allows retrieval or setting of the coded character set (CCS) associated with a Sequential (SSDF) or Relative (DSDF) file.

I18NLIB service routines

Allow direct access from COBOL compiler programs to the I18N services provided by I18NLIB.

PROGRAM COLLATING SEQUENCE

Defines the comparison of character strings. (This is a standard COBOL capability; it is not internationalized.)

4.2.1. DISP-I18N USAGE Clause

Function

When a DISP-I18N data item is explicitly used in a comparison such as IF, EVALUATE, or SEARCH, the compiler generates a call to the STRING_COLLATE service routine to perform the string comparison according to the rules in the current locale. Using this data type can eliminate the need to use the UCOB$STRCOLL call. The USAGE clause has no other effect on the data item.

Restriction

This clause can only be specified on elementary data items that are declared as either alphabetic (PIC A) or alphanumeric (PIC X).

General Format

```
USAGE IS DISP-I18N
```

where the USAGE and IS keywords are optional.

Results

The following results occur if an operand is declared as USAGE DISP-I18N, when this operand is part of a comparison that explicitly uses two operands:

1. The COBOL compiler generates a call internally to the STRING_COLLATE service routine.
2. The STRING_COLLATE service routine then performs the comparison.

UCOB$STRCOLL and the USAGE IS DISP-I18N Clause

The UCOB$STRCOLL call compares the two strings by using the collation rules in the current locale. (To set the current locale for a program, use the SETLOCALE service routine.) If UCOB$STRCOLL is used, the following conditions apply to the USAGE IS DISP-I18N clause:

- It is ignored for class conditions (such as ALPHABETIC and NUMERIC). The data item is treated the same as a USAGE IS DISPLAY data item, which means that only 7-bit U.S. ASCII alphabetic and numeric characters are recognized.
- It is flagged as non-standard if the COBOL compiler calls include the LEVEL/STANDARD keyword option.

Sample Code: USAGE IS DISP-I18N Clause (COBOL Compiler)

This code example shows how to use this clause to (1) compare two strings using the COBOL IF statement and (2) perform the same comparison using the STRING_COLLATE service routine. It is assumed that PROGRAM COLLATING SEQUENCE is not in effect.

```
 01 string-var1 PIC x(5) USAGE IS DISP-I18N.
 01 string-var2 PIC x(5) USAGE IS DISP-I18N.
 01 string-var3 PIC x(5).
 01 string-var4 PIC x(5).
 01 collation-result  PIC S9(10) BINARY.
        88 col-before VALUE -1.
        88 col-equal VALUE 0.
        88 col-after VALUE 1.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
      .
      .
      .
     MOVE 'ABCDE' TO string-var1.
     MOVE 'abcde' TO string-var2.
     MOVE 'abcde' TO string-var3.
     MOVE 'ABCDE' TO string-var4.
*
* Two strings are compared using the COBOL IF
* statement.   Because the two strings are defined
* with USAGE IS DISP-I18N, the compiler generates an
* internal call to STRING_COLLATE.
*
```

```
* The service routine determines the collation order
* using the rules determined by LC_COLLATE setting.
* Different locales will cause the strings to collate
* differently.
*
* In this routine, "ABCDE" will collate before "abcde"
* if LC_COLLATE is set to the value "en_US.ASCII",
* which corresponds to the English language as spoken
* in the U.S. using the ASCII codeset.  If LC_COLLATE
* indicates some other locale, then the two strings
* might collate in a different order.
*
     IF string-var1 = string-var2
       DISPLAY 'ABCDE collates equally with abcde'
     END-IF.
*
* The preceding COBOL statement is equivalent to the
* following direct call to STRING_COLLATE:
*
     CALL 'UCOB$STRCOLL' USING isl-error-status, string-var1,
        string-var2, collation-result.
*
* Check return status.
*
     IF NOT isl-successful
       IF isl-warning
       DISPLAY 'Warning ' isl-error-num ' received'
       ELSE
       DISPLAY 'Major error ' isl-error-num ' received'
       END-IF
       STOP RUN
     END-IF
*
* Check the result.
*
     IF col-equal
       DISPLAY 'ABCDE collates equally with abcde'
     END-IF.
*
* If only one of the data items in the IF statement is
* defined with USAGE IS DISP-I18N, the compiler generates
* an internal call to the UCOB$STRCOLL service routine.
* For example, string-var1 has the USAGE IS DISP-I18N
* clause; string-var3 does not.
*
     IF string-var1 = string-var3
       DISPLAY 'ABCDE collates equally with abcde'
     END-IF.
*
* If neither of the data items in the comparison has a
* USAGE IS DISP-I18N clause, the compiler generates a
* comparison of the strings using machine instructions.
```

```
*
* In the following example, the result would be false
* because the binary comparison using machine
* instructions would determine that the characters
* in the strings are not the same, due to upper- and
* lowercasing differences.
*
     IF string-var3 = string-var4
       DISPLAY 'ABCDE collates equally with abcde'
     END-IF.
*
     STOP RUN.
```

4.2.2. BYTE-I18N USAGE Clause

Function

The BYTE-I18N usage clause declares the data item as a string of bits. Typically, the USAGE IS BYTE-I18N clause identifies a data item containing the value returned by the STRING_TRANSFORM service routine.

Requirements

- The USAGE IS BYTE-I18N clause can only be specified on elementary data items declared as alphanumeric (PIC X).
- Data items defined with USAGE IS BYTE-I18N cannot be used in any arithmetic or string manipulation statements. When used in a comparison, the data item with USAGE IS BYTE-I18N is treated as a bit string; no conversion is done. If one data value is shorter than the other, the shorter value will be zero-padded to the right.
- A data item defined with USAGE IS BYTE-I18N can be used as the receiver in a MOVE statement only if the sender is another BYTE-I18N data item or one of the figurative constants LOW-VALUES or HIGH-VALUES. If the data item is longer than the sender, the receiver is padded with binary zeros on the right. If the data item is shorter than the sender, the value is truncated to the length of the receiver.
- A data item defined with USAGE IS BYTE-I18N can be used as the sender in a MOVE statement only if the receiver is also a BYTE-I18N data item.

General Format

```
USAGE IS BYTE-I18N
```

where the USAGE and IS keywords are optional.

STRING_TRANSFORM: Sample Code

You can use the following code routines to convert a character string into an ordering key.

- Creating Two Transformed Strings
- Using UCOB$STRXFRM With a SORT Call
- Using a Transformed Key to Create an Indexed Sequential File Used by PCIOS

4.2.2.1. Creating Two Transformed Strings

This code example shows how to use STRING_TRANSFORM to convert a character string to an ordering key. In this routine, it is assumed that PROGRAM COLLATING SEQUENCE is not in effect.

```
 *
 * Call UCOB$STRXFRM twice to produce two transformed
 * strings.
 *
 01  string-var1 PIC x(5).
 01  buffer-var1 PIC x(30) USAGE BYTE-I18N.
 01  buffer-var-len1 PIC 9(10) BINARY.
 01  string-var2 PIC x(5).
 01  buffer-var2 PIC x(30) USAGE BYTE-I18N.
 01  buffer-var-len2 PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
       .
       .
       .
     MOVE 'ABCDE' TO string-var1.
*
* Call UCOB$STRXFRM with these parameters:
*      Return status
*      String to be transformed
*      Transformed string set by UCOB$STRXFRM
*      Length of transformed string set by UCOB$STRXFRM
*
     CALL 'ucob$strxfrm' using isl-error-status, string-var1,
        buffer-var1, buffer-var-len1.
*
* Check return status.
*
     IF NOT isl-successful
        IF isl-warning
           DISPLAY 'Warning ' isl-error-num ' received'
        ELSE
           DISPLAY 'Major error ' isl-error-num ' received'
        END-IF
        STOP RUN
     END-IF.
     MOVE 'abcde' TO string-var2.
*
* Call UCOB$STRXFRM with these parameters:
*      Return status
*      String to be transformed
*      Transformed string set by UCOB$STRXFRM
*      Length of transformed string set by UCOB$STRXFRM
```

```
*
     CALL 'ucob$strxfrm' using isl-error-status, string-var2,
        buffer-var2, buffer-var-len2.
*
* Check return status.
*
     IF NOT isl-successful
        IF isl-warning
           DISPLAY 'Warning ' isl-error-num ' received'
        ELSE
           DISPLAY 'Major error ' isl-error-num ' received.
        END-IF
        STOP RUN
     END-IF.
*
* Compare the two strings.
*
     IF buffer-var1 < buffer-var2
        DISPLAY 'ABCDE collates before abcde'
     ELSE
        IF buffer-var1 = buffer-var2
           DISPLAY 'ABCDE collates equally with abcde'
        ELSE
           DISPLAY 'ABCDE collates after abcde'
        END-IF
     END-IF.
     STOP RUN.
```

4.2.2.2. Using UCOB$STRXFRM with a SORT Call

This code example shows how to use UCOB$STRXFRM in conjunction with a call to SORT. In this routine, it is assumed that PROGRAM COLLATING SEQUENCE is not in effect.

```
 FD  in-file.
 01  r1.
     02 in-employee-name PIC x(5).
     02 in-record-data PIC x(50).
 FD  out-file.
 01  r2.
     02 out-employee-name PIC x(5).
     02 out-record-data PIC x(50).
 SD  sort-file.
 01  r3.
     02 transformed-key PIC x(30) USAGE BYTE-I18N.
     02 r3rec.
         03 sort-employee-name PIC x(5).
         03 sort-record-data PIC x(50).
          .
          .
          .
```

```
 01  buffer-var-len PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
          .
          .
          .
*
* Sort a file by transforming the key into binary form
* before calling SORT.
*
     OPEN INPUT in-file.
     OPEN OUTPUT out-file.
     SORT sort-file ON ASCENDING KEY transformed-key.
         INPUT PROCEDURE in-proc1 THRU in-proc2
         OUTPUT PROCEDURE out-proc1 THRU out-proc2.
     CLOSE in-file out-file.
     STOP RUN.
 IN-PROC1.
*
* Read input file and move data to the sort record.
*
     READ in-file AT END GO TO in-proc2.
     MOVE r1 TO r3rec.
     MOVE SPACES TO transformed-key.
*
* Transform the key to a binary value that can be
* passed to SORT.
*
     CALL 'ucob$strxfrm' USING isl-error-status, sort-employee-
name,
        transformed-key, buffer-var-len.
     RELEASE r3.
     GO TO IN-PROC1.
 IN-PROC2.
     EXIT.
 OUT-PROC1.
*
* Get each record back from SORT, strip off the binary
* key, and write to the output file.
*
     RETURN sort-file AT END GO TO out-proc2.
     MOVE r3rec TO r2.
     WRITE r2.
     GO TO OUT-PROC1.
 OUT-PROC2.
     EXIT.
```

4.2.2.3. Using Transformed Key to Create Indexed Sequential File Used by PCIOS

This code example shows how to use a transformed key to create an indexed sequential file to be used by PCIOS. This key is stored in each record in the file.

```
              FILE-CONTROL.
                SELECT f1 ASSIGN TO DISC
                  ORGANIZATION INDEXED
                  RECORD KEY IS transformed-key
                  ACCESS SEQUENTIAL
              DATA DIVISION.
              FILE SECTION.
              FD f1.
              BLOCK CONTAINS 1 RECORDS.
              RECORD CONTAINS 135 CHARACTERS.
              01 rec.
                 02 transformed-key PIC x(30).
                 02 keydata PIC x(5).
                 02 record-data PIC x(100).
              WORKING-STORAGE SECTION.
              *
              * Copy in the declaration of isl-error-status.
              *
              COPY isl-err-copy.
              01 keylen PIC 9(10) BINARY.
              PROCEDURE DIVISION.
              BEGIN.
                  OPEN OUTPUT f1.
              *
              * Fill in fields in the record to be written to the file.
              *
                       MOVE ALL 'A' TO record-data.
                       MOVE 'ABCDE' TO keydata.
                       MOVE SPACES TO transformed-key.
              *
              * Transform the key into a form that can be used by PCIOS.
              *
                       CALL 'ucob$strxfrm' USING isl-error-status,
  keydata,
                          transformed-key, keylen.
                       WRITE rec INVALID KEY STOP RUN.
                       CLOSE f1.
```

4.2.3. SORT and MERGE Statements

Function

The SORT and MERGE statements support culturally sensitive collation. The following conditions apply:

- If a sort key is defined with the USAGE IS DISP-I18N clause, the SORT/MERGE operation for that key is based on the current locale. (An application program sets the locale by using the SETLOCALE service routine.)
- If a data item defined using USAGE IS BYTE-I18N is used as a sort key, the SORT/MERGE operation treats it as an unsigned binary sortable key that is, a U key type.

Requirements

- All file and data input to SORT and MERGE statements must have the same locale and the same CCS (CODE TYPE).
- Any file created as output from SORT and MERGE statements must have the same locale and the same CCS as the input file or data.

SORT Call and Data Items Defined with USAGE IS DISP-I18N: Sample Code

This code example shows how to use this call to define data items with USAGE IS DISP-I18N. These items are used in conjunction with a call to SORT. In this routine, it assumed that PROGRAM COLLATING SEQUENCE is not in effect.

```
 FD  in-file.
 01  r1.
        02 in-employee-name PIC x(5) USAGE IS DISP-I18N
        02 in-record-data PIC x(50)
 FD  out-file.
 01  r2.
        02 out-employee-name PIC x(5) USAGE IS DISP-I18N
        02 out-record-data PIC x(50)
 SD  sort-file.
 01  r3.
     02 sort-employee-name PIC x(5) USAGE IS DISP-I18N
     02 sort-record-data PIC x(50)
     .
     .
     .

*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
*
* Declare variables for SETLOCALE calls.
*
 01  locale PIC X(20).
 01  length PIC 9(10) BINARY.
    .
    .
    .
*
* Establish the current environment by calling SETLOCALE.
* The following routine sets the collation processing to
* follow the rules for the Norwegian language.
```

```
*
     CALL 'ucob$setloc' USING isl-error-status, 'LC_COLLATE',
        'no_NO.8859-1', locale, length.
*
* Sort a file with a DISP-I18N sort key.
*
     SORT sort-file ON ASCENDING KEY sort-employee-name
        USING in-file
        GIVING out-file.
     STOP RUN.
```

SORT Call and UCOB$STRXFRM: Sample Code

This code example shows how to use the STRING_TRANSFORM service routine in conjunction with SORT. In this example, it is assumed that PROGRAM COLLATING SEQUENCE is not in effect.

```
 FD in-file.
 01 r1.
    02 in-employee-name PIC x(5).
    02 in-record-data PIC x(50).
 FD out-file.
 01 r2.
        02 out-employee-name PIC x(5).
        02 out-record-data PIC x(50).
  SD sort-file.
  01 r3.
        02 transformed-key PIC x(30).
        02 r3rec.
             03 sort-employee-name PIC x(5).
             03 sort-record-data PIC x(50).
          .
          .
          .
 01 buffer-var-len PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
          .
          .
          .

*
* Sort a file by transforming the key into binary form
* before calling SORT.
*

     OPEN INPUT in-file.
     OPEN OUTPUT out-file.
     SORT sort-file ON ASCENDING KEY transformed-key
        INPUT PROCEDURE in-proc1 THRU in-proc2
```

```
           OUTPUT PROCEDURE out-proc1 THRU out-proc2.
        CLOSE in-file out-file.
        STOP RUN.
 IN-PROC1.
*
* Read input file and move data to the sort record.
*
        READ in-file AT END GO TO in-proc2.
        MOVE r1 to r3rec.
        MOVE LOW-VALUES to transformed-key.
*
* Transform the key to a binary value that can be passed
* to SORT.
*
        CALL 'ucob$strxfrm' USING isl-error-status,
           sort-employee-name, transformed-key,
           buffer-var-len.
        RELEASE r3.
        GO TO IN-PROC1.
 IN-PROC2.
        EXIT.
 OUT-PROC1.
*
* Get each record back from SORT, strip off the binary
* key, and write to the output file.
*
        RETURN sort-file AT END GO TO out-proc2.
        MOVE r3rec TO r2.
        WRITE r2.
        GO TO OUT-PROC1.
 OUT-PROC2.
        EXIT.
```

4.2.4. LOCALE Clause on SELECT Statement

Function

The LOCALE clause on the SELECT statement tells an application which locale will be used for processing the file data of an indexed (MSAM) file being read or created. The Locale Field is in the header block of the indexed file. Knowing the locale allows the application program to perform such operations as

- Tagging the file with the locale under which it was created
- Setting or retrieving the locale information for the indexed file that is being processed

General Format

```
LOCALE IS data-name
```

Input Values

LOCALE

is the clause allowed for files that (1) specify the ORGANIZATION INDEXED clause and (2) use the following as implement or names in the SELECT clause: DISC and A-SHARED-FILE.

IS

is an optional keyword.

data-name

must be defined in the Working-Storage Section as a character data item described as PIC x (alphanumeric category). The maximum size for *data-name* is 16 characters. Trailing spaces in the value are ignored. If *data-name* does not contain an initial value (VALUE clause) for a locale, the COBOL compiler initialize it to the locale name "POSIX". The locale name contained in *data-name* is case-sensitive, and must represent a valid locale for the system.

UCOB$STRXFRM and PCIOS/SFS Files

PCIOS/SFS files are handled somewhat differently than MSAM files; the locale of the data being processed is not considered. Instead, the application program must use UCOB$STRXFRM to generate binary orderable keys for PCIOS/SFS to use. The format of the transformed key varies, depending on the locale used to create the ordering key. The locale for creating the key is stored in the file header, so that the application program knows which locale to use when processing the file.

PCIOS: Sample Code for MSAM Files

This code example shows how to identify the locale for data in an MSAM file.

```
*
* Declare the file.
*
     SELECT fl ASSIGN TO DISC
               ORGANIZATION IS INDEXED
               LOCALE IS locale-name.
*
* Declare the variable locale-name.
*
 01 locale-name    PIC X(16).
          .
          .
          .
*
* Get locale when opening existing file.
*
     OPEN INPUT fl.
     IF locale-name NOT = 'en_US.ASCII'
         DISPLAY 'LOCALE is not en_US.ASCII' UPON PRINTER.
```

```
           STOP RUN.
*
* Establish locale when opening new file for output.
*
     MOVE "en_US.ASCII" TO locale-name.
     OPEN OUTPUT fl.
          .
          .
          .
```

Rules for Processing Files (COBOL Compiler)

Table 4–1. Rules for Processing Files

If the OPEN type is	And the file is	The following results occur:
INPUT	Available	Normal OPEN; LOCALE is set.
	Not available	OPEN is unsuccessful; LOCALE is not set.
INPUT (optional file)	Available	Normal OPEN; LOCALE is set.
	Not available	Normal OPEN; First read causes "at end" condition; LOCALE is not set.
I-O	Available	Normal OPEN; LOCALE is set.
	Not available	OPEN is unsuccessful; LOCALE is not set.
I-O (optional file)	Available	Normal OPEN; LOCALE is set.
	Not available	OPEN causes the file to be created; LOCALE in file is set to the LOCALE value.
OUTPUT	Available	Normal OPEN; File contains no records; LOCALE in file is set to the LOCALE value.
	Not available	OPEN causes the file to be created; LOCALE in file is set to the LOCALE value.
EXTEND	Available	Normal OPEN; LOCALE is set.
	Not available	OPEN unsuccessful; LOCALE is not set.

Table 4–1. Rules for Processing Files

If the OPEN type is	And the file is	The following results occur:
EXTEND (optional file)	Available	Normal OPEN; LOCALE is set.
	Not Available	OPEN causes the file to be created; LOCALE in file is set to the LOCALE value.

PCIOS: Sample Code for SSDF Files

This code example shows how to identify the CCS for data in an SSDF file.

```
*
* Declare the file.
*
     SELECT fl ASSIGN TO DISC
               CODE TYPE IS file-ccs.
*
* Declare the variable file-ccs.
*
 01 file-ccs    PIC 99 BINARY.
          .
          .
          .
* Get CCS when opening existing file.  It will be placed
* in the CODE TYPE variable when the file is opened.
*
     OPEN INPUT fl.
     IF file-ccs NOT = 35
        DISPLAY 'Code Type is not ISO 8859-1' UPON PRINTER.
        STOP RUN.
*
* Establish CCS when opening new file for output.  The
* CCS of the file will be set to the value of the variable
* specified in the CODE TYPE clause of the file being opened.
*
     MOVE 35 TO file-ccs.
     OPEN OUTPUT fl.
```

or

```
     CALL 'UCOB$CNM2CNB' USING isl-status, 'ISO8859-1',
       ccs-number-format, file-ccs.
     OPEN OUTPUT fl.
```

4.2.5. CODE TYPE Clause on SELECT Statement

Function

The CODE TYPE clause on the SELECT statement enables an application to determine the code type of the file being read or created. The Code Type Field is in the PCIOS/SFS and SAR$ header block of a file.

General Format

```
CODE TYPE IS data-name
```

Input Values

CODE TYPE

is the clause allowed for files that use the following implementer names in the SELECT clause: A-SHARED-FILE, ANS74-PRINTER, CARD-PUNCH, CARD-READER, DISC, PRINTER, and TAPE.

The following conditions apply to TAPE:

- If the implementer name is TAPE, it cannot be an IBM tape (RECORDING MODE clause specified).
- If TAPE is specified, the CODE TYPE clause is ignored unless the file is assigned to a mass storage device at execution time (device independence).
- If the file is assigned to this device, it is treated as if it were originally assigned to DISC in the source program.

IS

is an optional keyword.

data-name

must be defined in the Working-Storage Section as an unsigned numeric field described as PIC 99 (USAGE BINARY). The following conditions apply to data names:

- If *data-name* does not contain an initial value (VALUE clause) for a locale, the COBOL compiler initialize it to 1 (ASCII character set).
- When an application reads from a PCIOS/SFS file or reads input from a symbiont file, the value of *data-name* in the CODE TYPE clause is set to the CCS of the images being read.
- When an application creates a PCIOS/SFS file or writes output to a symbiont file, the value of *data-name* in the CODE TYPE clause is used to create the Code Type Indicator in the header.
- If the CODE TYPE clause is not specified, the code type of the currently active locale is stored.

Rules for Processing Files

See Table 4–1 for the rules for processing files.

Sample Code

This code example shows how to identify the CCS for data in an SSDF file.

```
*
* Declare the file.
*
     SELECT fl ASSIGN TO DISC
               CODE TYPE IS file-ccs.
*
* Declare the variable file-ccs.
*
 01 file-ccs    PIC 99 BINARY.
          .
          .
          .
* Get CCS when opening existing file.  It will be placed
* in the CODE TYPE variable when the file is opened.
*
     OPEN INPUT fl.
     IF file-ccs NOT = 35
        DISPLAY 'Code Type is not ISO 8859-1' UPON PRINTER.
        STOP RUN.
*
* Establish CCS when opening new file for output.  The
* CCS of the file will be set to the value of the variable
* specified in the CODE TYPE clause of the file being opened.
*
     MOVE 35 TO file-ccs.
     OPEN OUTPUT fl.
```

or

```
     CALL 'UCOB$CNM2CNB' USING isl-status, 'ISO8859-1',
       ccs-number-format, file-ccs.
     OPEN OUTPUT fl.
```

4.2.6. ALPHABET Clause and PROGRAM COLLATING SEQUENCE Clause

Function

The COBOL language contain two clauses that provide some internationalization support: the ALPHABET clause and the PROGRAM COLLATING SEQUENCE clause.

- The ALPHABET clause is a part of the SPECIAL-NAMES paragraph. This clause can be used to define a single ordering sequence for the characters in an alphabet.

- The PROGRAM COLLATING SEQUENCE clause is a part of the OBJECT-COMPUTER paragraph. It identifies the alphabet to be used for the program.

Constraints

The PROGRAM COLLATING SEQUENCE and ALPHABET clauses are used in conjunction with the SORT and MERGE statements to determine the collating order of characters in the alphabet. Because this operation is a one-dimensional ordering, it cannot fully represent the ordering requirements for languages with these specifications:

- Diacritical marks are weighted in order of importance from the end of the word to the beginning of the word.
- Two characters in the word are treated as a single collation item.
- One character in the word is treated as two characters for collation purposes.

4.3. I18NLIB COBOL Data Structures

The following subsections describe the COBOL data structure defined and used by the I18NLIB service routines. It also identifies the COPY procedures that define the data items. All I18NLIB COBOL COPY procedures are installed in the system procedure file SYSLIB*PROC$.

4.3.1. Error Structure: ISL-ERR-COPY

The ISL_ERR-COPY procedure defines the error return status.

```
01  isl-error-status.
    02  isl-severity          pic 1(3)  usage  binary-1.
        88  isl-successful                    value 0.
        88  isl-informational                 value 2.
        88  isl-warning                       value 3.
        88  isl-major-error                   value 7.
    02  isl-component-id      pic 1(15) usage  binary-1.
    02  isl-error-num         pic 1(18) usage  binary-1.
        88  isl-ok                            value 0.
        88  isl-not-ok                        value 1 thru 99.
```

4.3.2. Error Statuses: ISL-ERROR-STATUS

These error statuses can be returned as a numeric value in the *isl-error-status* parameter by the I18NLIB service routines called by the COBOL Compiler. For more information about each error, refer to the individual service routines.

Table 4–2. Sample Error Status List

Error Code	Severity Level	Name of Error Status
0	0	OK
3	7	BUFFER TOO SHORT

Table 4–2. Sample Error Status List

Error Code	Severity Level	Name of Error Status
4	7	FROM CCS NOT DEFINED
5	7	TO CCS NOT DEFINED
7	7	ENCODING NOT SUPPORTED
8	7	ENCODING OF TO BUFFER FAILED
13	7	BAD CHAR
14	7	BAD DESCRIPTOR
15	7	TOO MANY PARAMETERS
16	7	TOO FEW PARAMETERS
17	7	OVERLAPPING OPERANDS
20	7	EV NAME INVALID CHAR
21	7	EV VALUE INVALID CHAR
22	7	EV AREA FULL
23	7	EV NAME ILLEGAL FOR TIP
24	7	EV NAME NOT FOUND
25	7	EV VALUE TOO LONG
26	7	LOCALE NOT FOUND
27	7	NOT A UNISYS INTERMEDIATE CCS
28	7	INVALID CATEGORY
29	7	CATEGORY NOT SET
30	7	CORRUPTED TABLE
31	7	CCS NOT FOUND
32	7	FROM CCS EQUALS TO CCS
33	7	CCS BANK FULL
36	7	STRING 1 TOO LONG
37	7	STRING 2 TOO LONG
38	7	PUT$ENV ERROR
39	7	GET$ENV ERROR
40	7	CCS NUMBER NOT UNIQUE
41	7	ASTORE READ ERROR
42	7	ASTORE WRITE ERROR
43	7	CREATE BANK ERROR
44	7	CHECKPOINT RESTART ERROR

Table 4–2. Sample Error Status List

Error Code	Severity Level	Name of Error Status
45	7	CCS NUMBER UNDEFINED FORMAT
50	7	INVALID LOCALE ITEM
51	7	INVALID CONVERSION SPECIFICATION
52	7	INVALID ARGUMENT VALUES
53	7	TOO FEW ARGUMENTS
54	7	DESCRIPTOR DOES NOT MATCH INPUT CHR
55	3	MODIFIED CONVERSION SPECIFICATION NOT SUPPORTED
56	7	INVALID DATE/TIME VALUE
57	7	INVALID NMNB PACKET IDENTIFIER
58	7	INVALID NAME LENGTH IN NMNB PACKET
59	7	SYSTEM$TIME ERROR

4.3.3. Date/Time Structures (STRFTIME and STRPTIME)

The date/time structure for these two service routines is a nine-word integer packet.

Table 4–3. STRFTIME and STRPTIME Date/Time Structures

This word	Contains or indicates	And must be within this range
0	Number of seconds after the minute. The upper limit is 60 for handling "leap seconds."	[0, 60]
1	Number of minutes after the hour.	[0, 59]
2	Number of hours since midnight.	[0, 23]
3	Day of the month.	[1, 31]
4	Number of months since January. If the current month is January, this number is 0.	[0, 11]
5	Number of years since 1900 A -1 in this field indicates the year 1899. A 100 in this word indicates the year 2000.	[-1899, 8099]
6	Number of days since Sunday. If the current day is Sunday, this number is 0.	[0, 6]
7	Number of days since January 1. If the current day is January 1, this number is 0.	[0,365]

Table 4–3. STRFTIME and STRPTIME Date/Time Structures

This word	Contains or indicates	And must be within this range
8	Status of Daylight Savings Time positive, DST in effect; 0, DST not in effect, and negative, no DST information available	language-specific

4.4. I18NLIB COBOL Service Routines

The following subsections describe the COBOL Compiler Bindings to the I18NLIB service routines.

4.4.1. CCS_NAME_TO_CCS_NUM Service Routine

Function Call

UCOB$CNM2CNB

Description

CCS_NAME_TO_CCS_NUM transforms a coded character set (CCS) name to its corresponding CCS number.

General Format

```
CALL 'UCOB$CNM2CNB' USING isl-error-status, ccs-name,
        ccs-number-format, ccs-number.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

ccs-name

is the name of the CCS. This name is case-sensitive and must correspond to (1) a CCS_NAME entity in the Repository for ClearPath OS 2200 and (2) a CCS that is active on the system.

ccs-number-format

is a one-word binary item declared as PIC 9(10) BINARY, which indicates the kind of CCS number desired.

CCS Number = the Unisys CCS number used for CCS_TO_CCS transliteration.

CCS ISO Number = the ISO CCS number used for interoperability

CCS SDF Number = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records

ccs-number

is the corresponding CCS number. If the service routine returns an error status other than OK, the value for *ccs-number* is undefined.

Error Statuses

CCS_NAME_TO_CCS_NUM can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
31	7	CCS Not Found
45	7	CCS Number Format Undefined Error

Sample Code

This code example shows how to use CCS_NAME_TO_CCS_NUM to transform a CCS name to its corresponding CCS number.

```
 01  ccs-name PIC x(9).
 01  ccs-number PIC 9(10) BINARY.
 01  ccs-number-format PIC 9(10) BINARY.
         88 ccs-format VALUE 0.
         88 iso-format VALUE 1.
         88 sdf-format VALUE 2.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
          .
          .
          .
     MOVE 'ISO8859-1' TO ccs-name.
     MOVE 0 to ccs-number-format.
*
* Call UCOB$CNM2CNB with these parameters:
*      return status, ccs-name, ccs-number-format,
*      ccs-number
*
     CALL 'ucob$cnm2cnb' using isl-error-status, ccs-name,
        ccs-number-format, ccs-number.
*
```

```
* Check return status.
*
      IF NOT isl-successful
         IF isl-warning
            DISPLAY 'Warning ' isl-error-num ' received'
         ELSE
            DISPLAY 'Major error ' isl-error-num ' received'
         END-IF
      END-IF.
              .
              .
              .
```

4.4.2. CCS_NUM_TO_CCS_NAME Service Routine

Function Call

UCOB$CNB2CNM

Description

CCS_NUM_TO_CCS_NAME transforms a coded character set (CCS) number to its corresponding CCS name.

General Format

```
CALL 'UCOB$CNB2CNM' USING isl-error-status, ccs-number-format,
            ccs-number, ccs-name, name-length.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

ccs-number-format

is a one-word binary item declared as PIC 9(10) BINARY, which indicates the kind of CCS number provided.

CCS Number = the Unisys CCS number used for CCS_TO_CCS transliteration

CCS ISO Number = the ISO CCS number used for interoperability

CCS SDF Number = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records

ccs-number

is the CCS number. The CCS must participate in a CCS-TO-CCS conversion that is active on the system.

ccs-name

is the corresponding name of the CCS. This name is case-sensitive and corresponds to a CCS_NAME entity in the Repository for ClearPath OS 2200. If the service routine returns an error status other than OK, the value for *ccs-name* is undefined.

name-length

is the character length of *ccs-name.* If the service routine returns an error status other than OK, the value for *name-length* is undefined.

Error Statuses

CCS_NUM_TO_CCS_NAME can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
31	7	CCS Not Found
40	7	CCS Number Not Unique
45	7	CCS Number Format Undefined Error

Sample Code

This code example shows how to use CCS_NUM_TO_CCS_NAME to transform a CCS number to its corresponding CCS name.

```
 01  ccs-name PIC x(12).
 01  ccs-number PIC 9(10) BINARY.
 01  ccs-number-format PIC 9(10) BINARY.
         88 ccs-format VALUE 0.
         88 iso-format VALUE 1.
         88 sdf-format VALUE 2.
 01  name-length PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
          .
          .
          .
     MOVE 35 TO ccs-number.
     MOVE 0 TO ccs-number-format.
*
* Call UCOB$CNB2CNM with these parameters:
*      return status, ccs-number-format, ccs-name,
*      name-length
```

```
*
     CALL 'ucob$cnb2cnm' using isl-error-status,
         ccs-number-format, ccs-name, name-length.
*
* Check return status.
*
     IF NOT isl-successful
         IF isl-warning
            DISPLAY 'Warning ' isl-error-num ' received'
         ELSE
            DISPLAY 'Major error ' isl-error-num ' received'
         END-IF
     END-IF.
              .
              .
              .
```

4.4.3. CCS_TO_CCS Service Routine

Function Call

UCOB$CCS2CCS

Description

OPEN_CCS, CCS_TO_CCS, and CLOSE_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

General Format

```
CALL 'UCOB$CCS2CCS' USING isl-error-status, cv-descriptor,
          from-string, to-string, to-length.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

from-ccs

is a one-word binary item defined as PIC 9(10) BINARY. This item contains the Unisys CCS number of the *from-string*.

to-ccs

is a one-word binary item defined as PIC 9(10) BINARY. This item contains the Unisys CCS number of the *to-string*.

cv-descriptor

is the conversion descriptor, the internal identifier returned by the service routine for the transliteration tables. It must be defined as PIC 9(10) BINARY.

from-string

is the character variable (data name) or literal that the service routine is to transliterate. The data name must be defined as a string of one or more characters.

to-string

is the character variable that will contain the transliterated character string.

to-length

is the length of transliterated string in bytes. It must be defined as PIC 9(10) BINARY.

Error Statuses

The string transliteration service routines can return the following error statuses in the *isl-error-num* field:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
4	7	FROM-CCS Not Defined
5	7	TO-CCS Not Defined
13	7	Bad Character
14	7	Bad Descriptor
15	7	Too Many Parameters
16	7	Too Few Parameters
17	7	Overlapping Operands
30	7	Corrupted Table
31	7	CSS Not Found
32	7	FROM-CCS Equals TO-CCS

Sample Code

This code example shows how to use OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS to transliterate a character string from one CCS to another.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare variables.
*
```

```
 01  ccs-646fr PIC 9(10) BINARY VALUE 19.
 01  ccs-8859-1 PIC 9(10) BINARY VALUE 35.
 01  to-buffer PIC X(20).
 01  descriptor PIC 9(10) BINARY.
 01  to-length PIC 9(10) BINARY.
                .
                .
                .
*
* The following example shows the step-by-step
* procedure call (OPEN_CCS_TO_CCS, CCS_TO_CCS,
* CLOSE_CCS_TO_CCS) for transliterating characters
* 'résumé' in ISO 646 French (CCS #19) to ISO 8859.1
* (CCS #35).
*
     CALL 'UCOB$OPENCCS' USING isl-error-status, ccs-646fr,
        ccs-8859-1, descriptor.
     IF NOT isl-successful
        DISPLAY isl-error-num
        STOP RUN.
*
* Call CCS_TO_CCS service routine.
* Then call CLOSE_CCS_TO_CCS.
*
     CALL 'UCOB$CCS2CCS' USING isl-error-status, descriptor,
        'résumé',to-buffer,to-length.
     IF NOT isl-successful
        DISPLAY isl-error-num.
*
     CALL 'UCOB$CLOSCCS' USING isl-error-status, descriptor.
                    .
                    .
                    .
```

4.4.4. CLOSE_CCS Service Routine

Function Calls

UCOB$CLOSCCS

Description

OPEN_CCS, CCS_TO_CCS, and CLOSE_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

General Format

```
CALL 'UCOB$CLOSCCS' USING isl-error-status, cv-descriptor.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

cv-descriptor

is the conversion descriptor, the internal identifier returned by the service routine for the transliteration tables. It must be defined as PIC 9(10) BINARY.

Error Statuses

The string transliteration service routines can return the following error statuses in the *isl-error-num* field:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
4	7	FROM-CCS Not Defined
5	7	TO-CCS Not Defined
13	7	Bad Character
14	7	Bad Descriptor
15	7	Too Many Parameters
16	7	Too Few Parameters
17	7	Overlapping Operands
30	7	Corrupted Table
31	7	CSS Not Found
32	7	FROM-CCS Equals TO-CCS

Sample Code

This code example shows how to use OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS to transliterate a character string from one CCS to another.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare variables.
*
 01  ccs-646fr PIC 9(10) BINARY VALUE 19.
 01  ccs-8859-1 PIC 9(10) BINARY VALUE 35.
 01  to-buffer PIC X(20).
 01  descriptor PIC 9(10) BINARY.
```

```
 01  to-length PIC 9(10) BINARY.
                .
                .
                .
*
* The following example shows the step-by-step
* procedure call (OPEN_CCS_TO_CCS, CCS_TO_CCS,
* CLOSE_CCS_TO_CCS) for transliterating characters
* 'résumé' in ISO 646 French (CCS #19) to ISO 8859.1
* (CCS #35).
*
     CALL 'UCOB$OPENCCS' USING isl-error-status, ccs-646fr,
        ccs-8859-1, descriptor.
     IF NOT isl-successful
        DISPLAY isl-error-num
        STOP RUN.
*
* Call CCS_TO_CCS service routine.
* Then call CLOSE_CCS_TO_CCS.
*
     CALL 'UCOB$CCS2CCS' USING isl-error-status, descriptor,
        'résumé',to-buffer,to-length.
     IF NOT isl-successful
        DISPLAY isl-error-num.
*
     CALL 'UCOB$CLOSCCS' USING isl-error-status, descriptor.
                    .
                    .
                    .
```

4.4.5. CONVERT_FORM Service Routine

Function Call

UCOB$CNVRTFM

Description

CONVERT_FORM performs these tasks:

- Converts a character from its current representation in one portion of a character set to its representation in another portion of the same character set. There is always a one-to-one correspondence in the transliteration that occurs.
- CONVERT_FORM uses the current value of the locale in the LC_CTYPE category to determine which transliteration table to use. The calling program must ensure that LC_TYPE is set to a locale that contains the appropriate transliteration table.

General Format

```
CALL 'UCOB$CNVRTFM' USING isl-error-status, string-in,
          string-out, convert-dir.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string-in

is the character variable that contains the string to be converted. This variable must be defined as one or more characters long.

string-out

is the character variable that contains the converted character string. This variable must be at least as large as *string-in*.

convert-dir

is a one-word binary item declared as PIC 9(10) BINARY. This item indicates the direction for the conversion. The possible values are

0 = transliterate to the 7-bit range

1 = transliterate to the 8-bit range

Error Statuses

CONVERT_FORM can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
15	7	Too Many Parameters
16	7	Too Few Parameters
17	7	Overlapping Operands
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
27	7	Not a Unisys Intermediate Locale
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error

Error Code	Severity Code	Description
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use CONVERT_FORM to change the encoding of a character string.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg PIC X(5).
 01  outbuff PIC X(12).
 01  direction PIC 9(10) BINARY.
     88 to-7-bit-range VALUE 0.
     88 to-8-bit-range VALUE 1.
*
* The internal octal representation of 'süß' in
* ISO 646 German is
*    s = 163
*    ü = 175
*    ß = 176
*
         .
         .
         .
     MOVE 'süß' TO strg.
     SET to-8-bit-range TO TRUE.
*
* Call CONVERT_FORM to convert the string to its 8-bit
* representation in ISO 8859-1.  The locale that will be
* used for the transliteration will be determined by the
* setting of LC_CTYPE.  The locale must be defined as a
* Unisys intermediate locale for Germany.
*
     CALL 'ucob$cnvrtfm' USING isl-error-status, strg, outbuff,
         direction.
     IF NOT isl-successful
        DISPLAY 'Conversion failed'.
*
* The output buffer contains the internal octal
* representation of 'süß' in ISO 8859-1.
*    s = 163
*    ü = 374
```

```
*     ß = 337
*
```

4.4.6. GETENV Service Routine

Function Call

UCOB$GETENV

Description

GETENV retrieves the current value of the environment variable set at the run level. These variables are initially set by the default values from the user profile.

General Format

```
CALL 'UCOB$GETENV' USING isl-error-status, env-name,
            env-value, env-value-len.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

env-name

is the name of the environment variable defined as a character variable or literal.

env-value

is a character variable that contains the value that is returned from UCOB$GETENV. If the operation fails, this value is undefined.

env-value-len

is a one-word binary item defined as PIC 9(10) BINARY. This item contains the character length of the returned value. If the operation fails, this length is undefined.

Error Statuses

GETENV can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
20	7	EV Name Contains an Invalid Character
21	7	EV Value Contains an Invalid Character
24	3	EV Name Not Found
39	7	GET$ENV Error

Sample Code

This code example shows how to use GETENV to retrieve the value of environment variables (EV) at the run level. These routines retrieve the value of LC_COLLATE. The following declarations are used in both examples:

env-var = environment variable

env-val = EV value returned from GETENV

env-val-len = length of EV value returned

```
 01  env-var PIC x(10).
 01  env-val PIC x(16).
 01  env-val-len PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
```

Example 1

```
*
* LC_COLLATE is the environment variable.
*
     MOVE 'LC_COLLATE' TO env-var.
*
* Call GETENV with these parameters:
*   Return status
*   Environment variable
*   EV value set by GETENV
*   Length of EV value set by GETENV
*
     CALL 'ucob$getenv' USING isl-error-status, env-var,
        env-val, env-val-len.
*
* Check return status.
*
     IF NOT isl-successful
        IF isl-warning
           DISPLAY 'Warning ' isl-error-num ' received'
        ELSE
           DISPLAY 'Major error ' isl-error-num ' received'
        END-IF
        STOP RUN
     END-IF.
*
* Display LC_COLLATE value if return status = OK.
*
     DISPLAY 'Current LC_COLLATE value is ' env-val.
```

Example 2

```
*
* Call GETENV with these parameters:
*   Return status
*   EV name passed as a literal
*   EV value set by GETENV
*   Length of EV value set by GETENV
*
     CALL 'ucob$getenv' USING isl-error-status, 'LC_COLLATE',
        env-val, env-val-len.
*
* Display LC_COLLATE value.
*
     DISPLAY 'Current LC_COLLATE value is ' env-val.
```

4.4.7. GETENVSYS Service Routine

Function Call

UCOB$GETENVS

Description

GETENVSYS retrieves the current value of the environment variable set at the system level.

General Format

```
CALL 'UCOB$GETENVS' USING isl-error-status, env-name,
env-value, env-value-len.
```

Exception for Trailing Spaces

These service routines allow the caller to indicate if trailing spaces in the environment variable value are significant or not.

Parameters

isl-error-status

is the error structure returned by the service routine.

env-name

is the name of the environment variable defined as a character variable or literal.

env-value

is a character variable that contains the value that is returned from UCOB$GETENVS. If the operation fails, this value is undefined.

env-value-len

is a one-word binary item defined as PIC 9(10) BINARY. This item contains the character length of the returned value. If the operation fails, this length is undefined.

Error Statuses

GETENVSYS can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
20	7	EV Name Contains an Invalid Character
21	7	EV Value Contains an Invalid Character
24	3	EV Name Not Found
39	7	GET$ENV Error

Sample Code

This code example shows how to use GETENV to retrieve the value of environment variables (EV) at the system level. These routines retrieve the value of LC_COLLATE. The following declarations are used in both examples:

env-var = environment variable

env-val = EV value returned from GETENVSYS

env-val-len = length of EV value returned

```
 01  env-var PIC x(10).
 01  env-val PIC x(16).
 01  env-val-len PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
```

Example 1

```
*
* LC_COLLATE is the environment variable.
*
     MOVE 'LC_COLLATE' TO env-var.
*
* Call GETENVSYS with these parameters:
*   Return status
*   Environment variable
*   EV value set by GETENVSYS
*   Length of EV value set by GETENVSYS
```

```
*
     CALL 'ucob$getenvs' USING isl-error-status, env-var,
        env-val, env-val-len.
*
* Check return status.
*
     IF NOT isl-successful
        IF isl-warning
           DISPLAY 'Warning ' isl-error-num ' received'
        ELSE
           DISPLAY 'Major error ' isl-error-num ' received'
        END-IF
        STOP RUN
     END-IF.
*
* Display LC_COLLATE value if return status = OK.
*
     DISPLAY 'Current LC_COLLATE value is ' env-val.
```

Example 2

```
*
* LC_COLLATE is the environment variable.
*
* Call GETENVSYS with these parameters:
*   Return status
*   EV name passed as a literal
*   EV value set by GETENVSYS
*   Length of EV value set by GETENVSYS
*
     CALL 'ucob$getenvs'  USING isl-error-status, 'LC_COLLATE',
        env-val, env-val-len.
*
* Display LC_COLLATE value.
*
     DISPLAY 'Current LC_COLLATE value is ' env-val.
```

4.4.8. ISALNUM Service Routine

Function Call

UCOB$ISALNUM

Description

ISALNUM determines if all the characters in the string are in the alphanumeric character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the alpha and digit categories in the locale definition file.

Relation to COBOL IS ALPHANUMERIC Clause

ISALNUM does not alter the way the COBOL IS ALPHANUMERIC clause operates. This clause continues to use the ASCII-coded character set or the values for the ALPHABET clause specified in the program.

General Format

```
CALL 'UCOB$ISALNUM' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error

Error Code	Severity Code	Description
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISALNUM to determine if all characters of the input string are alphanumeric.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88  isl-condition-false VALUE 0.
         88  isl-condition-true VALUE 1.
                .
                .
                .
*
* ISALNUM is called to determine if the identifier
* passed contains all alphanumeric characters.  An
* error status is returned in isl-error-status.
*
     CALL 'ucob$isalnum' USING isl-error-status, strg-var,
         isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is alphanumeric'
     ELSE
        IF isl-successful
           DISPLAY 'String is not alphanumeric'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
       .
       .
       .
```

4.4.9. ISALPHA Service Routine

Function Call

UCOB$ISALPHA

Description

ISALPHA determines if all the characters in the string are in the alphabetic character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the upper, lower, and letter categories in the locale definition file.

Relation to COBOL IS ALPHABETIC Clause

ISALPHA does not alter the way the COBOL IS ALPHABETIC clause operates. This clause continues to use the ASCII-coded character set or the values for the ALPHABET clause specified in the program.

General Format

```
CALL 'UCOB$ISALPHA' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISALPHA to determine if all characters of the input string are alphabetic.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88  isl-condition-false VALUE 0.
         88  isl-condition-true VALUE 1.
                 .
                 .
                 .
*
* ISALPHA is called to determine if the identifier
* passed contains all alphanumeric characters.  An
* error status is returned in isl-error-status.
```

```
*
     CALL 'ucob$isalpha' USING isl-error-status, strg-var,
        isl-condition.
     IF  isl-condition-true
        DISPLAY 'String is alphanumeric'
     ELSE
        IF  isl-successful
           DISPLAY 'String is not alphanumeric'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
        .
        .
        .
```

4.4.10. ISCNTRL Service Routine

Function Call

UCOB$ISCNTRL

Description

ISCNTRL determines if all the characters in the string are in the control character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the cntrl category in the locale definition file.

General Format

```
CALL 'UCOB$ISCNTRL' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISCNTRL to determine if all characters of the input string are control characters.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88  isl-condition-false VALUE 0.
         88  isl-condition-true VALUE 1.
                 .
                 .
                 .
```

```
*
* ISCNTRL is called to determine if the identifier
* passed contains all control characters.  An error
* status is returned in isl-error-status.
*
     CALL 'ucob$iscntrl' USING isl-error-status, strg-var,
         isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is control characters'
     ELSE
        IF isl-successful
           DISPLAY 'String is not all control characters'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
         .
         .
         .
```

4.4.11. ISDIGIT Service Routine

Function Call

UCOB$ISDIGIT

Description

ISDIGIT determines if all the characters in the string are decimal digits in the numeric character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the digit category in the locale definition file.

General Format

```
CALL 'UCOB$ISDIGIT' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISDIGIT to determine if all the characters of the input string are numeric (decimal digits).

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
```

```
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88 isl-condition-false VALUE 0.
         88 isl-condition-true VALUE 1.
                 .
                 .
                 .
*
* ISDIGIT is called to determine if the identifier
* passed contains all numeric characters.  An error
* status is returned in isl-error-status.
*
     CALL 'ucob$isdigit' USING isl-error-status, strg-var,
        isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is numeric'
     ELSE
        IF isl-successful
           DISPLAY 'String is not numeric'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
       .
       .
       .
```

4.4.12. ISGRAPH Service Routine

Function Call

UCOB$ISGRAPH

Description

ISGRAPH determines if all the characters in the string are in the graphic character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the graph category in the locale definition file.

General Format

```
CALL 'UCOB$ISGRAPH' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISGRAPH to determine if all the characters of the input string are graphic characters.

```
*
* Copy in the declaration of the return error status.
*
```

```
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
        88 isl-condition-false VALUE 0.
        88 isl-condition-true VALUE 1.
                  .
                  .
                  .
*
* ISGRAPH is called to determine if the identifier
* passed contains all graphical characters.  An error
* status is returned in isl-error-status.
*
     CALL 'ucob$isgraph' USING isl-error-status, strg-var,
        isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is graphical'
     ELSE
        IF isl-successful
           DISPLAY 'String is not graphical'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
       .
       .
       .
```

4.4.13. ISLOWER Service Routine

Function Call

UCOB$ISLOWER

Description

ISLOWER determines if all the characters in the string are in the lowercase character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the lower category in the locale definition file.

General Format

```
CALL 'UCOB$ISLOWER' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISLOWER to determine if the characters of the input string are lowercase and alphabetic.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02 var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88 isl-condition-false VALUE 0.
         88 isl-condition-true VALUE 1.
               .
               .
               .
*
* ISLOWER is called to determine if the identifier
* passed contains all lower case characters.  An error
* status is returned in isl-error-status.
*
     CALL 'ucob$islower'  USING isl-error-status, strg-var,
        isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is lowercase'
     ELSE
        IF isl-successful
           DISPLAY 'String is not lowercase'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
       .
       .
       .
```

4.4.14. ISPRINT Service Routine

Function Call

UCOB$ISPRINT

Description

ISPRINT determines if all the characters in the string are in the printable character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the print category in the locale definition file.

General Format

```
CALL 'UCOB$ISPRINT' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error

Error Code	Severity Code	Description
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISPRINT to determine if the characters of the input string are printable.

```
*
* Copy in the declaration of the return error status.
*
    COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02 var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88 isl-condition-false VALUE 0.
         88 isl-condition-true VALUE 1.
                  .
                  .
                  .
*
* ISPRINT is called to determine if the identifier
* passed contains all printable characters.  An error
* status is returned in isl-error-status.
*
     CALL 'ucob$isprint' USING isl-error-status, strg-var,
        isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is printable'
     ELSE
        IF isl-successful
           DISPLAY 'String is not printable'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
                  .
                  .
                  .
```

4.4.15. ISPUNCT Service Routine

Function Call

UCOB$ISPUNCT

Description

ISPUNCT determines if all the characters in the string are in the punctuation character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the punct category in the locale definition file.

General Format

```
CALL 'UCOB$ISPUNCT' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters

Error Code	Severity Code	Description
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example tells how to use ISPUNCT to determine if the characters of the input string are punctuation characters.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88 isl-condition-false VALUE 0.
         88 isl-condition-true VALUE 1.
                    .
                    .
                    .
*
* ISPUNCT is called to determine if the identifier
* passed contains all punctuation characters.  An
* error status is returned in isl-error-status.
*
     CALL 'ucob$ispunct'  USING isl-error-status, strg-var,
        isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is punctuation'
     ELSE
```

```
      IF isl-successful
         DISPLAY 'String is not punctuation'
      ELSE
         DISPLAY 'An error occurred'
      END-IF
   END-IF.
      .
      .
      .
```

4.4.16. ISSPACE Service Routine

Function Call

UCOB$ISSPACE

Description

ISSPACE determines if all the characters in the string are in the white space character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the space category in the locale definition file.

General Format

```
CALL 'UCOB$ISSPACE' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISSPACE to determine if the characters of the input string are white space characters.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88 isl-condition-false VALUE 0.
         88 isl-condition-true VALUE 1.
                     .
                     .
                     .
*
* ISSPACE is called to determine if the identifier
* passed contains all white space characters.  An
* error status is returned in isl-error-status.
```

```
*
     CALL 'ucob$isspace' USING isl-error-status, strg-var,
        isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is white space characters'
     ELSE
        IF isl-successful
          DISPLAY 'String is not white space characters'
        ELSE
          DISPLAY 'An error occurred'
        END-IF
     END-IF.
        .
        .
        .
```

4.4.17. ISUPPER Service Routine

Function Call

UCOB$ISUPPER

Description

ISUPPER determines if all the characters in the string are in the uppercase character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the upper category in the locale definition file.

General Format

```
CALL 'UCOB$ISUPPER' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISUPPER to determine if the characters of the input string are uppercase and alphabetic.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88 isl-condition-false VALUE 0.
        88 isl-condition-true VALUE 1.
                   .
```

```
                    .
                    .
*
* ISUPPER is called to determine if the identifier
* passed contains all upper case characters.  An
* error status is returned in isl-error-status.
*
     CALL 'ucob$isupper' USING isl-error-status, strg-var,
        isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is uppercase'
     ELSE
        IF isl-successful
           DISPLAY 'String is not uppercase'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
       .
       .
       .
```

4.4.18. ISXDIGIT Service Routine

Function Call

UCOB$ISXDIG

Description

ISXDIGIT determines if all the characters in the string are in the hexadecimal digit (xdigit) character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the xdigit category in the locale definition file.

General Format

```
CALL 'UCOB$ISXDIG' USING isl-error-status, string, result.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string

is the character variable or literal to be checked by the service routine. This variable or literal must be defined as a string of one or more characters.

result

is a one-word integer described as PIC 9(10) BINARY, which indicates whether all the characters in the string are in the correct character class for the service routine.

1 = All the characters are in the correct character class.

0 = One or more of the characters are not in the correct character class.

If 0 is returned, check *isl-error-status* to determine if an error occurred.

Error Statuses

The character class service routines can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use ISXDIGIT to determine if the characters of the input string are hexadecimal digits.

```
*
* Copy in the declaration of the return error status.
*
```

```
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var.
     02  var1 PIC X(5).
 01  isl-condition PIC 9(10) BINARY.
         88 isl-condition-false VALUE 0.
         88 isl-condition-true VALUE 1.
                   .
                   .
                   .
*
* ISXDIGIT is called to determine if the identifier
* passed contains all hexadecimal characters.  An
* error status is returned in isl-error-status.
*
     CALL 'ucob$isxdig' USING isl-error-status, strg-var,
         isl-condition.
*
     IF isl-condition-true
        DISPLAY 'String is hexadecimal'
     ELSE
        IF isl-successful
           DISPLAY 'String is not hexadecimal'
        ELSE
           DISPLAY 'An error occurred'
        END-IF
     END-IF.
       .
       .
       .
```

4.4.19. LOCALE_NAME_TO_CCS_NUM Service Routine

Function Call

UCOB$LNM2CNB

Description

LOCALE_NAME_TO_CCS_NUM returns the coded character set (CCS) number associated with the locale name.

General Format

```
CALL 'UCOB$LNM2CNB' USING isl-error-status, locale-name,
          ccs-number-format, ccs-number.
```

Parameters

isl-error-status

is the error structure returned by the service routine

locale-name

is the name of the locale. This name is required by the SETLOCALE service routine. It is case-sensitive and must correspond to (1) a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and (2) a locale that is active on the system.

ccs-number-format

is a one-word binary item declared as PIC 9(10) BINARY, which indicates the kind of CCS number provided.

CCS Number = the Unisys CCS number used for CCS_TO_CCS transliteration

CCS ISO Number = the ISO CCS number used for interoperability

CCS SDF Number = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records

ccs-number

is the number of the CCS associated with the locale. If the service routine returns an error status other than OK, the value for *ccs-number* is undefined.

Error Statuses

LOCALE_NAME_TO_CCS_NUM can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
26	7	Locale Not Found
45	7	CCS Number Format Undefined Error

Sample Code

This code example shows how to use LOCALE_NAME_TO_CCS_NUM to return the CCS number associated with the locale name.

```
01  locale-name PIC x(12).
01  ccs-number-format PIC 9(10) BINARY.
```

```
         88 ccs-format VALUE 0.
         88 iso-format VALUE 1.
         88 sdf-format VALUE 2.
 01  ccs-number PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
         .
         .
         .
     MOVE 'fr_FR.8859-1' TO locale-name.
     MOVE 0 TO ccs-number-format.
*
* Call UCOB$LNM2CNB with these parameters:
*      return status, locale-name, ccs-number-format,
*      ccs-number
*
     CALL 'ucob$lnm2cnb' using isl-error-status, locale-name,
        ccs-number-format, ccs-number.
*
* Check return status.
*
     IF NOT isl-successful
        IF isl-warning
           DISPLAY 'Warning ' isl-error-num ' received'
        ELSE
           DISPLAY 'Major error ' isl-error-num ' received'
        END-IF
     END-IF.
        .
        .
        .
```

4.4.20. LOCALE_NAME_TO_LOCALE_NUM Service Routine

Function Call

UCOB$LNM2LNB

Description

LOCALE_NAME_TO_LOCALE_NUM transforms a locale name to its corresponding locale number.

General Format

```
CALL 'UCOB$LNM2LNB' USING isl-error-status, locale-name,
            locale-number.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

locale-name

is the name of the locale. This name is required by the SETLOCALE service routine. It is case-sensitive and must correspond to (1) a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and (2) a locale that is active on the system.

locale-number

is the corresponding locale number. If the service routine returns an error status other than OK, the value for locale-number is undefined.

Error Statuses

LOCALE_NAME_TO_LOCALE_NUM can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
26	7	Locale Not Found

Sample Code

This code example shows how to use LOCALE_NAME_TO_LOCALE_NUM to transform a locale name to its corresponding locale number.

```
 01  locale-name PIC x(12).
 01  locale-number PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
          .
          .
          .
     MOVE 'fr_FR.8859-1' TO locale-name.
*
* Call UCOB$LNM2LNB with these parameters:
*      return status, locale-name, locale-number
*
     CALL 'ucob$lnm2lnb' using isl-error-status, locale-name,
        locale-number.
```

```
*
* Check return status.
*
     IF NOT isl-successful
        IF isl-warning
           DISPLAY 'Warning ' isl-error-num ' received'
        ELSE
           DISPLAY 'Major error ' isl-error-num ' received'
        END-IF
     END-IF.
         .
         .
         .
```

4.4.21. LOCALE_NUM_TO_CCS_NUM Service Routine

Function Call

UCOB$LNB2CNB

Description

LOCALE_NUM_TO_CCS_NUM returns the coded character set (CCS) number associated with the locale number.

General Format

```
CALL 'UCOB$LNB2CNB' USING isl-error-status, locale-number,
            ccs-number-format, ccs-number.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

locale-number

is the number of a locale that is active on the system.

ccs-number-format

is a one-word binary item declared as PIC 9(10) BINARY, which indicates the kind of CCS number provided.

CCS Number = the Unisys CCS number used for CCS_TO_CCS transliteration

CCS ISO Number = the ISO CCS number used for interoperability

CCS SDF Number = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records

ccs-number

is the number of the CCS associated with the locale. If the service routine returns an error status other than OK, the value for *ccs-number* is undefined.

Error Statuses

LOCALE_NUM_TO_CCS_NUM can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
26	7	Locale Not Found
45	7	CCS Number Format Undefined Error

Sample Code

This code example shows how to use LOCALE_NUM_TO_CCS_NUM to return the CCS number associated with the locale number.

```
 01  locale-number PIC 9(10) BINARY.
 01  ccs-number-format PIC 9(10) BINARY.
         88 ccs-format VALUE 0.
         88 iso-format VALUE 1.
         88 sdf-format VALUE 2.
 01  ccs-number PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
          .
          .
          .
     MOVE 6 TO locale-number.
     MOVE 0 to ccs-number-format.
*
* Call UCOB$LNB2CNB with these parameters:
*      return status, locale-number, ccs-number-format,
*      ccs-number
*
     CALL 'ucob$lnb2cnb' using isl-error-status, locale-number,
        ccs-number-format, ccs-number.
*
* Check return status.
*
     IF NOT isl-successful
```

```
      IF isl-warning
         DISPLAY 'Warning ' isl-error-num ' received'
      ELSE
         DISPLAY 'Major error ' isl-error-num ' received'
      END-IF
   END-IF.
    .
    .
    .
```

4.4.22. LOCALE_NUM_TO_LOCALE_NAME Service Routine

Function Call

UCOB$LNB2LNM

Description

LOCALE_NUM_TO_LOCALE_NAME transforms a locale number to its corresponding locale name.

General Format

```
CALL 'UCOB$LNB2LNM' USING isl-error-status, locale-number,
             locale-name, name-length.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

locale-number

is the number of a locale that must be active on the system.

locale-name

is the corresponding locale name. This name is case-sensitive and corresponds to a LOCALE_NAME entity in the Repository for ClearPath OS 2200. If the service routine returns an error status other than OK, the value for locale-name is undefined.

name-length

is the character length of locale-name. If the service routine returns an error status other than OK, the value for *name-length* is undefined.

Error Statuses

LOCALE_NUM_TO_LOCALE_NAME can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
15	7	Too Many Parameters
16	7	Too Few Parameters
26	7	Locale Not Found

Sample Code

This code example shows how LOCALE_NUM_TO_LOCALE_NAME is used to transform a locale number to its corresponding locale name.

```
 01  locale-name PIC x(12).
 01  locale-number PIC 9(10) BINARY.
 01  name-length PIC 9(10) BINARY.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
         .
         .
         .
     MOVE 6 TO locale-number.
*
* Call UCOB$LNB2LNM with these parameters:
*      return status, locale-number, locale-name,
*      name-length
*
     CALL 'ucob$lnb2lnm' using isl-error-status, locale-number,
        locale-name, name-length.
*
* Check return status.
*
     IF NOT isl-successful
         IF isl-warning
            DISPLAY 'Warning ' isl-error-num ' received'
         ELSE
            DISPLAY 'Major error ' isl-error-num ' received'
         END-IF
     END-IF.
       .
       .
       .
```

4.4.23. NAME_AND_NUMBER Packet Interface

Function Call

UCOB$NMNB

Description

The NAME_AND_NUMBER packet interface provides a general interface for obtaining name-and-number information for various locales and CCSs. The Virtual Machine for the Java™ Platform on ClearPath OS 2200 can use this interface to obtain this information, which also includes the number of bits in a character for the requested locale or CCS.

Note: *All locales have an associated CCS number. However, not all CCSs are associated with a locale.*

General Format

```
CALL 'UCOB$NMNB' USING isl-error-status, isl-nmnb-information-packet.
```

Error Statuses

NAME_AND_NUMBER can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
2	7	Invalid Packet Version
15	7	Too Many Parameters
16	7	Too Few Parameters
26	7	Locale Not Found
31	7	CCS Not Found
40	7	CCS Number Not Unique
45	7	CCS Number Format Undefined
57	7	Invalid NMNB Packet Identifier
58	7	Invalid Name Length in NMNB Packet

Information Packet Structure

See Section 5 for a diagram of the NAME_AND_NUMBER information packet.

List of Fields in the Information Packet

The following table lists the following:

- The fields associated with CCS and locale data within the I18NLIB NAME_AND_NUMBER information packet (ISL-NMNBIP).

- The input requirements needed for the CCS and locale names and numbers in order to receive valid results in *result-status* and the return packet.

Field Name	Identifier	Description and Valid Values	Input Requirements	Results Returned
identifier		The type of information passed in the packet. The valid integer values are 1=CCS number 2=CCS ISO number 3=CCS SDF number 4=Locale number 5=CCS name 6=Locale name None=Error situation		
char-bit-size		The number of bits in a character for a requested CCS or one associated with a requested locale. The number can be 6, 9 (single-byte character), or 18 (double-byte character).		
version		The version of the packet.		
number		A specific CCS number, CCS ISO number, CCS SDF number, or locale number. This number can be from 0 through 262, 143.		
CCS name length		The number of 9-bit bytes in a CCS name. The range can be from 1 through 16.		
CCS number	1	A CCS number or one associated with a requested locale. This number can be from 0 through 262,143.	version identifier number	char-bit-size ccs-number ccs-iso-number ccs-sdf number ccs-name-length ccs-name

Field Name	Identifier	Description and Valid Values	Input Requirements	Results Returned
CCS ISO number	2	A CCS ISO number or one associated with a requested locale. This number can be from 0 through 262,143.	version identifier number	char-bit-size ccs-number ccs-iso-number ccs-sdf-number ccs-name-length ccs-name
CCS SDF number	3	A CCS SDF number or one associated with a requested locale. This number can be from 0 through 262,143.	version identifier number	char-bit-size ccs-number ccs-iso-number ccs-sdf-number ccs-name-length ccs-name
CCS name	5	A CCS name containing any printable 8-bit character from ISO 8859.1 except a space (octal 040). This name may have to be - A four-word area that is left-justified and space-filled. - Case-sensitive and corresponding to a CCS_NAME entity in the Repository. - A CCS that is active on the system.	version identifier number	version identifier number
locale name length		The number of 9-bit bytes in a locale name. The range can be from 1 through 16.		

Field Name	Identifier	Description and Valid Values	Input Requirements	Results Returned
Locale number	4	This field contains a locale number associated with a locale name. The range can be from 0 to 262,143.	version identifier number	char-bit-size ccs-number ccs-iso-number ccs-sdf-number ccs-name-length ccs-name locale-name-length locale-number locale name
Locale name	6	A locale name containing any printable 8-bit character from ISO 8859-1 except a space (octal 040) or a slash (/) character (octal 057). This name may have to be - A four-word area that is left-justified and space-filled. - Case-sensitive and corresponding to a LOCALE_NAME entity in the Repository - A locale that is active on the system.	identifier locale-name-length locale-name	char-bit-size ccs-name-length ccs-name ccs-number ccs-iso-number ccs-sdf-number locale-name-length locale-number locale-name

The following code defines the structure of the NAME_AND_NUMBER information packet for internationalized OS 2200 applications that use the COBOL Compiler. This code is included in the SYSLIB*PROC$ file of the I18N Service Library.

```
01  isl-nmnb-information-packet.
    05  isl-nmnb-reserved-1          pic 1(18) binary-1.
    05  isl-nmnb-identifier          pic 1(06) binary-1.
    05  isl-nmnb-char-bit-size       pic 1(06) binary-1.
    05  isl-nmnb-version             pic 1(06) binary-1.
    05  isl-nmnb-number              pic 1(36) binary-1.
    05  isl-nmnb-ccs-name-length     pic 1(36) binary-1.
    05  isl-nmnb-ccs-number          pic 1(36) binary-1.
    05  isl-nmnb-ccs-iso-number      pic 1(36) binary-1.
    05  isl-nmnb-ccs-sdf-number      pic 1(36) binary-1.
    05  isl-nmnb-ccs-name            pic x(16) display.
    05  isl-nmnb-locale-name-length  pic 1(36) binary-1.
```

```
    05  isl-nmnb-locale-number       pic 1(36) binary-1.
    05  isl-nmnb-locale-name         pic x(16) display.
```

Sample Code

This code example shows how to use NAME_AND_NUMBER to obtain CCS and locale name-and-number information.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. UCISLNMNB01.
*  NAME_AND_NUMBER PACKET INTERFACE tests
*   for CCS Number input (identifier=1)
*  NORMAL CASE
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. UNISYS-2200.
 OBJECT-COMPUTER. UNISYS-2200.
 SPECIAL-NAMES.
     printer is prt.
 DATA DIVISION.
 FILE SECTION.
 WORKING-STORAGE SECTION.
*
*  COPY IN THE DECLARATION OF THE RETURN ERROR STATUS
*
        copy isl-err-copy.
*
*  COPY IN THE DECLARATION OF THE NAME_AND_NUMBER INFORMATION PKT
*
        copy isl-nmnb-information-packet.
*
*  DECLARE VARIABLES
*
 01  TEST-ITEM                   pic  9(5)  BINARY.
 01  LAST-ITEM                   pic  9(5)  BINARY  VALUE 7.
 01  TEST-DATA.
     02  TEST-DATA-ARRAY OCCURS 7.
         03  INPUT-DATA          pic  1(36) BINARY-1.
*
 PROCEDURE DIVISION.
 BEGIN.
     DISPLAY '  *** start UCISLNMNB01 ***' upon prt.
     PERFORM SET-INPUT-DATA.
     PERFORM CCS-NUMBER
        VARYING TEST-ITEM FROM 1 BY 1
        until TEST-ITEM > LAST-ITEM.
     DISPLAY '  *** end UCISLNMNB01 ***' upon prt.
     STOP RUN.
*
 CCS-NUMBER.
     MOVE LOW-VALUE TO isl-nmnb-information-packet.
     MOVE 0 TO isl-nmnb-reserved-1.
```

```
            move 1 to isl-nmnb-version.
            MOVE 1 TO isl-nmnb-identifier.
            MOVE INPUT-DATA(TEST-ITEM) TO isl-nmnb-number.
            CALL 'UCOB$NMNB' using isl-error-status,
                                   isl-nmnb-information-packet.
            IF isl-successful
              THEN
                DISPLAY '*** test for CCS Number = '
                        INPUT-DATA(TEST-ITEM) ' ***' upon prt
                DISPLAY 'OUTPUT : char-bit-size = '
                        isl-nmnb-char-bit-size upon prt
                DISPLAY 'OUTPUT : ccs-number = '
                        isl-nmnb-ccs-number upon prt
                DISPLAY 'OUTPUT : ccs-iso-number = '
                        isl-nmnb-ccs-iso-number upon prt
                DISPLAY 'OUTPUT : ccs-sdf-number = '
                        isl-nmnb-ccs-sdf-number upon prt
                DISPLAY 'OUTPUT : ccs-name-length = '
                        isl-nmnb-ccs-name-length upon prt
                DISPLAY 'OUTPUT : char-name = '
                        isl-nmnb-ccs-name       upon prt
              ELSE
                display 'ISL-NMNB returned an unexpected status' upon
prt
                display 'the status is ' isl-error-num upon prt
            END-IF.
       SET-INPUT-DATA.
            MOVE   0 TO INPUT-DATA(1).
            MOVE   1 TO INPUT-DATA(2).
            MOVE  35 TO INPUT-DATA(3).
            MOVE 160 TO INPUT-DATA(4).
            MOVE  20 TO INPUT-DATA(5).
            MOVE  61 TO INPUT-DATA(6).
            MOVE 195 TO INPUT-DATA(7).
```

4.4.24. NL_LANGINFO Service Routine

Function Call

UCOB$NLLANG

Description

NL_LANGINFO returns a string containing relevant information about the particular language or cultural area defined in the program's locale.

General Format

```
CALL 'UCOB$NLLANG' USING status, item, str-out, str-out-length.
```

Parameters

status

is the error structure returned by the service routine.

item

is a one-word binary item, declared as PIC 9(10) BINARY, indicating the locale item to retrieve. Values for these items are defined in the COPY PROC ISL-PKT-COPY.

str-out

is a character variable where the resulting string is written.

str-out-length

is a one-word binary item, declared as PIC 9(10) BINARY, that contains the number of characters in the resulting string.

Definitions for Locale Items Required by COBOL Compilers

The ISL-PKT-COPY/COBP element contains the following definitions for locale items required by NL_LANGINFO.

```
01  isl-codeset     PIC 9(10) BINARY VALUE 1.
01  isl-d-t-fmt     PIC 9(10) BINARY VALUE 2.
01  isl-d-fmt       PIC 9(10) BINARY VALUE 3.
01  isl-t-fmt       PIC 9(10) BINARY VALUE 4.
01  isl-t-fmt-ampm  PIC 9(10) BINARY VALUE 5.
01  isl-am-str      PIC 9(10) BINARY VALUE 6.
01  isl-pm-str      PIC 9(10) BINARY VALUE 7.
01  isl-day-1       PIC 9(10) BINARY VALUE 8.
01  isl-day-2       PIC 9(10) BINARY VALUE 9.
01  isl-day-3       PIC 9(10) BINARY VALUE 10.
01  isl-day-4       PIC 9(10) BINARY VALUE 11.
01  isl-day-5       PIC 9(10) BINARY VALUE 12.
01  isl-day-6       PIC 9(10) BINARY VALUE 13.
01  isl-day-7       PIC 9(10) BINARY VALUE 14.
01  isl-abday-1     PIC 9(10) BINARY VALUE 15.
01  isl-abday-2     PIC 9(10) BINARY VALUE 16.
01  isl-abday-3     PIC 9(10) BINARY VALUE 17.
01  isl-abday-4     PIC 9(10) BINARY VALUE 18.
01  isl-abday-5     PIC 9(10) BINARY VALUE 19.
01  isl-abday-6     PIC 9(10) BINARY VALUE 20.
01  isl-abday-7     PIC 9(10) BINARY VALUE 21.
01  isl-mon-1       PIC 9(10) BINARY VALUE 22.
01  isl-mon-2       PIC 9(10) BINARY VALUE 23.
01  isl-mon-3       PIC 9(10) BINARY VALUE 24.
01  isl-mon-4       PIC 9(10) BINARY VALUE 25.
01  isl-mon-5       PIC 9(10) BINARY VALUE 26.
01  isl-mon-6       PIC 9(10) BINARY VALUE 27.
01  isl-mon-7       PIC 9(10) BINARY VALUE 28.
```

```
01  isl-mon-8       PIC 9(10) BINARY VALUE 29.
01  isl-mon-9       PIC 9(10) BINARY VALUE 30.
01  isl-mon-10      PIC 9(10) BINARY VALUE 31.
01  isl-mon-11      PIC 9(10) BINARY VALUE 32.
01  isl-mon-12      PIC 9(10) BINARY VALUE 33.
01  isl-abmon-1     PIC 9(10) BINARY VALUE 34.
01  isl-abmon-2     PIC 9(10) BINARY VALUE 35.
01  isl-abmon-3     PIC 9(10) BINARY VALUE 36.
01  isl-abmon-4     PIC 9(10) BINARY VALUE 37.
01  isl-abmon-5     PIC 9(10) BINARY VALUE 38.
01  isl-abmon-6     PIC 9(10) BINARY VALUE 39.
01  isl-abmon-7     PIC 9(10) BINARY VALUE 40.
01  isl-abmon-8     PIC 9(10) BINARY VALUE 41.
01  isl-abmon-9     PIC 9(10) BINARY VALUE 42.
01  isl-abmon-10    PIC 9(10) BINARY VALUE 43.
01  isl-abmon-11    PIC 9(10) BINARY VALUE 44.
01  isl-abmon-12    PIC 9(10) BINARY VALUE 45.
01  isl-radixchar   PIC 9(10) BINARY VALUE 46.
01  isl-thousep     PIC 9(10) BINARY VALUE 47.
01  isl-yesexpr     PIC 9(10) BINARY VALUE 48.
01  isl-noexpr      PIC 9(10) BINARY VALUE 49.
01  isl-crncystr    PIC 9(10) BINARY VALUE 50.
```

Error Statuses

NL_LANGINFO can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error
50	7	Invalid Locale Item

Sample Code

This code example shows how to use NL_LANGINFO to return a string containing information about a particular language or cultural area defined in the program's locale. This routine illustrates a call to NL_LANGINFO to return the full name of the first month in the year, based on the POSIX locale, which specifies the minimal environment for C-language translation.

```
        .
        .
        .
 WORKING-STORAGE SECTION.
*
* Copy in the declaration of the return error status.
*
  COPY isl-err-copy.
*
* Copy in other structures (namely the item declarations).
*
  COPY isl-pkt-copy.
*
* Declare variables for the NL_LANGINFO call.
*
 01  item          PIC 9(10) BINARY.
 01  str-out       PIC X(15).
 01  str-out-len   PIC 9(10) BINARY.
*
 PROCEDURE DIVISION.
 BEGIN.
*
* Initialize parameters.
*
     MOVE isl-mon-1 TO item.
     MOVE SPACE TO str-out.
*
* Perform call to NL_LANGINFO.
*
     CALL 'ucob$nllang' USING isl-error-status, item,
                              str-out, str-out-len.
*
* Display result.
*
     IF isl-successful
        DISPLAY str-out
     END-IF.
     STOP RUN.
```

Assuming that the locale for LC_TIME is "POSIX", the following output is displayed:

```
January
```

4.4.25. OPEN_CCS Service Routine

Function Calls

UCOB$OPENCCS

Description

OPEN_CCS, CCS_TO_CCS, and CLOSE_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

General Format

```
CALL 'UCOB$OPENCCS' USING isl-error-status, from-ccs, to-ccs,
          cv-descriptor.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

from-ccs

is a one-word binary item defined as PIC 9(10) BINARY. This item contains the Unisys CCS number of the *from-string*.

to-ccs

is a one-word binary item defined as PIC 9(10) BINARY. This item contains the Unisys CCS number of the *to-string*.

cv-descriptor

is the conversion descriptor, the internal identifier returned by the service routine for the transliteration tables. It must be defined as PIC 9(10) BINARY.

Error Statuses

The string transliteration service routines can return the following error statuses in the *isl-error-num* field:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
4	7	FROM-CCS Not Defined
5	7	TO-CCS Not Defined
13	7	Bad Character
14	7	Bad Descriptor

Error Code	Severity Code	Description
15	7	Too Many Parameters
16	7	Too Few Parameters
17	7	Overlapping Operands
30	7	Corrupted Table
31	7	CSS Not Found
32	7	FROM-CCS Equals TO-CCS

Sample Code

This code example shows how to use OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS to transliterate a character string from one CCS to another.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare variables.
*
 01  ccs-646fr PIC 9(10) BINARY VALUE 19.
 01  ccs-8859-1 PIC 9(10) BINARY VALUE 35.
 01  to-buffer PIC X(20).
 01  descriptor PIC 9(10) BINARY.
 01  to-length PIC 9(10) BINARY.
                .
                .
                .
*
* The following example shows the step-by-step
* procedure call (OPEN_CCS_TO_CCS, CCS_TO_CCS,
* CLOSE_CCS_TO_CCS) for transliterating characters
* 'résumé' in ISO 646 French (CCS #19) to ISO 8859.1
* (CCS #35).
*
     CALL 'UCOB$OPENCCS' USING isl-error-status, ccs-646fr,
        ccs-8859-1, descriptor.
     IF NOT isl-successful
        DISPLAY isl-error-num
        STOP RUN.
*
* Call CCS_TO_CCS service routine.
* Then call CLOSE_CCS_TO_CCS.
*
     CALL 'UCOB$CCS2CCS' USING isl-error-status, descriptor,
        'résumé',to-buffer,to-length.
     IF NOT isl-successful
        DISPLAY isl-error-num.
```

```
*
     CALL 'UCOB$CLOSCCS' USING isl-error-status, descriptor.
                         .
                         .
                         .
```

4.4.26. PUTENV Service Routine

Function Call

UCOB$PUTENV

Description

PUTENV sets the current value of the environment variable for the run level.

General Format

```
CALL 'UCOB$PUTENV' USING isl-error-status,
          env-var-name-string, env-var-value-string.
```

Passing the Environment Variable Name and Value

The lengths of the environment variable name and value are passed to UCOB$PUTENV in the standard calling sequences, not by way of the PUTENV service routine.

Parameters

isl-error-status

is the error structure returned by the service routine.

env-var-name-string

is the name of an environment variable. Trailing spaces are ignored.

env-var-value-string

is the value to which the environment variable is to be set. Trailing spaces may be included in the string if terminated with binary zeroes.

The maximum length is 16 characters. In the TIP environment the maximum length of the LANG environment variable is 10 characters.

Error Statuses

PUTENV can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters

Error Code	Severity Code	Description
20	7	EV Name Invalid Character
21	7	EV Value Invalid Character
22	7	EV Area Full
23	7	EV Name Illegal for TIP
25	3	EV Value Too Long
38	7	PUT$ENV Error

Sample Code

These code examples show how to use PUTENV to set environment variables (EV) at the run level. These routines set the value of LC_COLLATE.

Example 1

```
      *
      * The following declarations are used in this example:
      *   env-var-name-string = EV name string
      *   env-var-value-string = EV value string
      *
       01  env-var-name-string PIC x(10).
       01  env-var-value-string PIC x(12).
      *
      * Copy in the declaration of isl-error-status.
      *
           COPY isl-err-copy.
                    .
                    .
                    .
      *
      * LC_COLLATE is the EV name;
      * fr_FR.8859-1 is the EV value.
      *
           MOVE 'LC_COLLATE' TO env-var-name-string.
           MOVE 'fr_FR.8859-1' TO env-var-value-string.
      *
      * Call PUTENV with these parameters:
      *   Return status
      *   EV name string
      *   EV value string
      *
           CALL 'ucob$putenv' USING isl-error-status,
              env-var-name-string, env-var-value-string.
      *
      * Check return status.
      *
           IF NOT isl-successful
              IF isl-warning
```

```
                DISPLAY 'Warning ' isl-error-num ' received'
             ELSE
                DISPLAY 'Major error ' isl-error-num ' received'
             END-IF
             STOP RUN
          END-IF.
Example 2
                  .
                  .
                  .
      *
      * Call PUTENV with these parameters:
      *     Return status
      *     EV name string passed as a literal
      *     EV value string passed as a literal
      *
           CALL 'ucob$putenv' USING isl-error-status, 'LC_ALL',
               'fr_FR.8859-1'.
                   .
                   .
                   .
```

4.4.27. SETLOCALE Service Routine

Function Call

UCOB$SETLOC

Description

SETLOCALE is used to set category values and query locale information for a user program.

The setting of category values is indeterminate when a user program begins execution. This program calls UCOB$SETLOC to set all or a portion of these values, as specified by the category and requested-locale fields.

SETLOCALE can query information about all or a portion of the program's locale. A query of the category LC_ALL returns the locale name for the LC_COLLATE category.

Once called by the program, SETLOCALE establishes the locale for the current activity. This operation includes such functions as specifying the locale name, and the length and storage location for this name.

General Format

```
CALL 'UCOB$SETLOC' USING isl-error-status, category,
       requested-locale, resulting-locale,
       resulting-locale-length.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

category

is a character variable or literal that specifies the portion of the program's locale to set or query. There are seven allowable categories: LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.

requested-locale

is a character variable that specifies the name of a locale, "_DEFAULT", or "_QUERY".

The maximum length for a locale name is 16 characters. When dealing with locale names, SETLOCALE is case-sensitive and ignores trailing spaces.

resulting-locale

is a character variable that specifies the locale name that is set or queried for in requested-locale.

If this operation is not successful, this value is undefined. The value may be the same for both requested-locale and resulting-locale.

resulting-locale-length

is a one-word binary item defined as PIC 9(10) BINARY. It contains the length, in characters, of the name in resulting-locale. If the SETLOCALE operation is not successful, this length is undefined.

Query Locale Value ("_QUERY")

This value is used to direct SETLOCALE to query the current locale of the program, according to the category value, and then return the name of the locale. If a locale name is not set for the requested category, it is set by using the heuristic method followed by the "_DEFAULT" locale value.

Error Statuses

See Appendix A for an explanation of the error codes.

SETLOCALE can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
15	7	Too Many Parameters

Error Code	Severity Code	Description
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
28	7	Invalid Category
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use SETLOCALE to establish a locale for an activity.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare variables for SETLOCALE calls.
*
 01  locale-name PIC X(20).
 01  length PIC 9(10) BINARY.
                .
                .
                .
*
* Set all the categories to their default values.
*
     CALL 'ucob$setloc' USING isl-error-status, 'LC_ALL',
     '_DEFAULT', locale-name, length.
*
* Do a query to determine the locale for category
* LC_COLLATE.
     CALL 'ucob$setloc' USING isl-error-status, 'LC_COLLATE',
     '_QUERY', locale-name, length.
*
* If LC_COLLATE's locale is not Norwegian, set it to
* Norwegian.
*
     IF locale-name(1:length) IS NOT = 'no_NO.8859-1'
```

```
      CALL 'ucob$setloc' USING isl-error-status, 'LC_COLLATE',
         'no_NO.8859-1', locale-name, length
    END-IF.
      .
      .
      .
```

4.4.28. STRFMON Service Routine

Function Call

UCOB$STRFMON

Description

STRFMON is used to convert numeric monetary values to a string representation based on a specified format and the localeterm information defined by the LC_MONETARY category.

General Format

```
CALL 'ucob$strfmon' USING status, fmt-str, arguments,
            str-out, str-out-length.
```

Parameters

status

is the error structure returned by the service routine.

fmt-str

is the string describing the format of the resulting output buffer. This string has two objects: plain characters and conversion specifications.

Plain Characters: Plain characters are copied unchanged to the output buffer.

Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

arguments

is a list of monetary values to be converted based on the format string. These values must be declared by using USAGE IS COMPUTATIONAL-2. STRFMON:

Examples of Declaring Arguments (COBOL Compiler)

```
01  arguments.
    02  argument1 USAGE IS COMPUTATIONAL-2.
    02  argument2 USAGE IS COMPUTATIONAL-2.
    02  argument3 USAGE IS COMPUTATIONAL-2.
```

or

```
01  arguments.
    02  argument OCCURS 3 TIMES COMPUTATIONAL-2.
```

str-out

is the output buffer where the resulting string is written.

str-out-length

is a one-word binary item, defined as PIC9(10) BINARY, that contains the number of characters in the resulting string.

Conversion Specification Table (STRFMON)

The following table shows the sequence of arguments in a conversion specification. Each conversion specification results in the fetching of zero or more sequential arguments that are converted and formatted.

- Each argument must be a double-precision floating point number.
- If excess arguments remain after the format string is exhausted, they are ignored.
- If the format string contains more conversion specifications than arguments, the results are unpredictable. With the C compiler, an error is returned.

Note: *The % character and the conversion character are required; all others are optional.*

Table 4–4. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
1	% character	Required argument
2	flags	Optional arguments
	=f	A numeric fill character. This character (1) must be representable in a single byte; (2) does not affect field width filling, which always uses the space character; and (3) is ignored unless left precision is specified. The default is the space character.
	^	Indicates that the currency amount should not be formatted with currency characters. If this flag is defined for the current locale, grouping characters are inserted.
	+ or (	Specifies the style for representing positive and negative currency amounts. Only one style can be specified. If +, the locale's equivalent of + and - are used. If (, negative amounts are enclosed in parentheses ( ). If neither style is specified, the default is +.
	!	Suppresses the currency symbol from output conversion.

Table 4–4. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
	=	Specifies the alignment. If it is present, all fields are left justified.
3	field width (w)	Optional argument. A decimal digit that specifies a minimum field width in bytes. If the - flag is specified, the result is left-justified. If the - flag is not specified, the result is right-justified. The default is zero (0).
4	left precision (#n)	Optional argument. A pound sign followed by a decimal digit string that specifies the maximum number of digits expected to be formatted to the left of the radix character. It can be used to (1) keep formatted output from multiple calls to STRFMON aligned in the same columns and (2) fill unused positions with a special character. • If more than n digit positions are required, this value is ignored, and the excess positions are filled with numeric fill characters (=f). • If the grouping has not been suppressed with the ^ flag and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit. • To ensure alignment, any characters appearing before or after the number in the formatted output (such as currency or sign symbols) are padded as necessary with space characters to make their positive and negative formats an equal length.
5	right precision (.p)	Optional argument. A period followed by a decimal digit that specifies the number of digits after the radix character. If the value is zero, no radix character appears . The amount being formatted is rounded to the specified number of digits before formatting occurs. If right precision is not included, a default specified by the current locale is used.
6	conversion character	Required argument
	i	This argument should use the locale's international currency format. USA example: `USD 1,234.56`
	n	This argument should use the locale's national currency format. USA example: `$1,234.56`
	%	This means no argument is converted. The entire conversion specification must be %%.

.

Return Values

On normal completion, the resulting string is placed in *str_out* and the number of characters written are placed in *str_out_length*. In any case, the error structure is always returned by the service routine.

Error Statuses

STRFMON can return the following error statuses in the *status* field:

Error Code	Severity Code	Description
0	7	OK (No Error)
3	7	Buffer Too Short
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error
51	7	Invalid Conversion Specification
52	7	Invalid Argument Value

Sample Code

This code example shows how to use STRFMON to format a monetary value.

```
        .
        .
        .
 WORKING-STORAGE SECTION.
*
* Copy in the declaration of the return error status
*
  COPY isl-err-copy.
*
* Declare variables for the STRFMON call.
*
 01  fmt-str       PIC X(35).
 01  str-out       PIC X(40).
 01  str-out-len   PIC 9(10) BINARY.
*
* Declare arguments for the STRFMON call.  They must be
```

```
* consecutive and declared with USAGE IS COMPUTATIONAL-2.
*
 01  arguments.
     02  argument1 USAGE IS COMPUTATIONAL-2.
     02  argument2 USAGE IS COMPUTATIONAL-2.
*
 PROCEDURE DIVISION.
 BEGIN.
*
* Initialize parameters.
*
     MOVE 'Cost is %=*#5n (%#5.0n rounded).' TO fmt-str.
     MOVE SPACE TO str-out.
     MOVE 123.45 TO argument1.
     MOVE 123.45 TO argument2.
*
* Perform call to STRFMON.
*
     CALL 'ucob$strfmon' USING isl-error-status, fmt-str,
                               arguments, str-out, str-out-len.
*
* Display result.
*
     IF isl-successful
        DISPLAY str-out
     END-IF.
     STOP RUN.
```

Assuming that the locale for LC_MONETARY is "POSIX", the following output is displayed:

```
Cost is $***123.45 ($  123 rounded).
```

4.4.29. STRFTIME Service Routine

Function Call

UCOB$STRFTM

Description

STRFTIME is used to convert date and time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

General Format

```
CALL 'ucob$strftim' USING status, fmt-str, isl-date-time,
                str-out, str-out-length.
```

Parameters

See 4.4.28 for a description of the Parameters.

Conversion Specification Table

Each conversion specification (CS) is replaced by the appropriate characters listed in this table. These characters are determined by the current locale and by the values contained in the date-and-time structure. If a CS does not correspond to any of the ones listed in this table, the behavior is undefined.

Table 4–5. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%a	Weekday name, abbreviated	abday	tm_wday
%A	Weekday name, full	day	tm_wday
%b	Month name, abbreviated	abmon	tm_mon
%B	Month name, full	mon	tm_mon
%c	Appropriate date/time representation	d_t_fmt	Varies by locale
%C	Century number. To get this number, the year is divided by 100 and truncated to an integer as a decimal [00-99].	n/a	tm_year
%d	Day of month as decimal [01,31]	n/a	tm_mday
%D	Equivalent to %m/%d/%y	n/a	tm_mon, tm_mday, and tm_year.
%e	Day of month as decimal [1,31]. A single digit is preceded by a space.	n/a	tm_mday
%F	Equivalent to %Y-%m-%d (the ISO8601:2000 standard date format)	n/a	tm_year, tm_mon, and tm_mday
%g	Replaced by the last 2 digits of the week-based year as a decimal number [00-99].	n/a	tm_year, tm_yday, and tm_mday
%G	Replaced by the week-based year and as a decimal number.	n/a	tm_year, tm_yday, tm_mday
%h	Month name, abbreviated	abmon	tm_mon
%H	Hour on 24-hour clock as decimal [00,23]	n/a	tm_hour
%I	Hour on 12-hour clock as decimal [01,12]	n/a	tm_hour
%j	Day of year as decimal [001,366]	n/a	tm_yday
%m	Month as decimal [01,12]	n/a	tm_mon
%M	Minute as decimal [00,59]	n/a	tm_min
%n	New line character (012)	n/a	n/a

Table 4–5. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%p	Equivalent for ante-meridiem (AM) and post-meridiem (PM)	am_pm	tm_hour
%r	Time in AM and PM notation; equivalent of %I:%M:%S %p in the POSIX locale	t_fmt_ampm	Refer to individual CSs.
%R	Time in 24-hour notation (%H:%M)	n/a	tm_hour and tm_min.
%S	Second as decimal [00,60]	n/a	tm_sec
%t	Tab character	n/a	n/a
%T	Time (%H:%M:%S)	n/a	tm_hour, tm_min, and tm_sec.
%u	Weekday as decimal [1,7]	n/a	tm_wday
%U	Week number of the year as decimal [00,53] with Sunday as day 1	n/a	tm_yday and tm_wday
%V	Week number of the year as decimal [01,53] with Monday as day 1 If the week containing January 1 has four or more days in the new year, then it is considered week 1. If is does not, then it is the last week of the previous year, and the next week is week 1.	n/a	tm_year, tm_mday, and tm_yday
%w	weekday as decimal [0,6] with 0 representing Sunday	n/a	tm_wday
%W	Week number of the year as decimal [00,53] with Monday as day 1. All days in a new year preceding the first Sunday are considered to be in week 0.	n/a	tm_wday and tm_yday
%x	Appropriate date representation	d_fmt	Varies by locale
%X	Appropriate time representation	t_fmt	Varies by locale
%y	Year without century as decimal [00,99]	n/a	tm_year
%Y	Year with century as decimal [1998]	n/a	tm_year
%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm) or by no characters if no timezone is determinable.	n/a	tm_isdst
%Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.	n/a	tm_isdst

Table 4–5. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%%	%	n/a	n/a

Return Values

Upon normal completion, the *str_out* string contains the proper converted characters, and the procedure returns the error status in *status*.

Error Statuses

STRFTIME can return the following error statuses in the *status* field:

Error Code	Severity Code	Description
0	7	OK (No Error)
3	7	Buffer Too Short
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error
51	7	Invalid Conversion Specification
56	7	Invalid Date/Time Value
59	7	SYSTEM$TIME error

Sample Code

This code example shows how to use STRFTIME to format a date value to a string representation based on locale en_US.8859-1 (English as spoken in the US).

```
        .
        .
        .
 WORKING-STORAGE SECTION.
*
 01  string-var       PIC X(12).
 01  buffer-var       PIC X(32).
 01  buffer-var-len   PIC 9(10 BINARY.
*
* Copy in the declaration for isl-error-status
*
  COPY isl-err-copy.
*
* Copy in the declaration for the isl-date-time structure.
*
  COPY isl-pkt-copy.
        .
        .
        .
     MOVE 'Today is %x.' to string-var.
     MOVE 7 TO tm-mon.
     MOVE 22 TO tm-mday.
     MOVE 95 TO tm-year.
     CALL 'ucob$strftim' using isl-error-status, string-var,
           isl-date-time, buffer-var, buffer-var-len.
*
* Check return status
*
     IF NOT isl-successful
        IF isl-warning
           DISPLAY 'WARNING ' isl-error-num ' RECEIVED'
        ELSE
           DISPLAY 'MAJOR ERROR ' isl-error-num ' RECEIVED'
        END-IF
     ELSE
        DISPLAY buffer-var
     END-IF.
     STOP-RUN.
```

Assuming the locale for LC_TIME is "en_US.8859-1", the following will be displayed:

```
Today is 08/22/95.
```

4.4.30. STRING_COLLATE Service Routine

Function Call

UCOB$STRCOLL

Description

STRING_COLLATE compares two character strings and returns their collating order. These strings can be up to 4096 characters long. The collating rules are determined by the current setting of the LC_COLLATE category.

General Format

```
CALL 'UCOB$STRCOLL' USING isl-error-status, in-string1,
          in-string2, collation-result
```

Parameters

isl-error-status

is the error structure returned by the service routine.

in-string1

is the first of the two strings to be compared.

in-string2

is the second of the two strings to be compared.

collation-result

is a one-word integer described as PIC S9(10) BINARY, which is set by STRING_COLLATE, and which indicates the result of the comparison. The possible values are

-1 = First string collates before second string.

0 = Strings collate equally.

1 = First string collates after second string.

Error Statuses

STRING_COLLATE can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found

Error Code	Severity Code	Description
30	7	Corrupted Table
36	7	String 1 Too Long
37	7	String 2 Too Long
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use STRING_COLLATE to compare two character strings and return their collating order. In this routine, it is assumed that PROGRAM COLLATING SEQUENCE is not in effect.

```
 01  string-var1 PIC x(5).
 01  string-var2 PIC x(5).
 01  collation-result PIC S9(10) BINARY.
        88  col-before VALUE -1.
        88  col-equal VALUE 0.
        88  col-after VALUE 1.
*
* Copy in the declaration of isl-error-status.
*
     COPY isl-err-copy.
       .
       .
       .
     MOVE 'ABCDE' TO string-var1.
     MOVE 'abcde' TO string-var2.
*
* Two strings are compared.  The service routine uses
* the rules determined by the LC_COLLATE setting to
* establish the collation.  Different locales will cause
* the strings to collate differently.
*
* In this routine, "ABCDE" will collate before "abcde"
* if LC_COLLATE is set to the value "en_US.ASCII",
* which corresponds to the English language as spoken
* in the U.S., using the ASCII codeset.
*
* If LC_COLLATE indicates some other locale, then the
* two strings might collate in a different order.
*
     CALL 'UCOB$STRCOLL' USING isl-error-status, string-var1,
```

```
           string-var2, collation-result.
*
* Check return status.
*
        IF NOT isl-successful
           IF isl-warning.
              DISPLAY 'Warning ' isl-error-num ' received'
           ELSE
              DISPLAY 'Major error ' isl-error-num ' received'
           END-IF
           STOP RUN
        END-IF.
*
* Check the result.
*
        IF col-before
           DISPLAY 'ABCDE collates before abcde'
        ELSE
           IF col-equal
             DISPLAY 'ABCDE collates equally with abcde'
          ELSE
             DISPLAY 'ABCDE collates after abcde'
          END-IF
        END-IF
        STOP RUN.
```

4.4.31. STRING_TRANSFORM Service Routine

Function Call

UCOB$STRXFRM

Description

STRING_TRANSFORM transforms a character string into an ordering key. This string can be up to 4096 characters long. The transformation rules are determined by the current setting of the LC_COLLATE category.

General Format

```
CALL 'UCOB$STRXFRM' USING isl-error-status, in-string,
           out-buffer, out-buffer-len
```

Handling Trailing Spaces

Any trailing spaces in the string variables that are passed as input to these service routines will be truncated.

Placing a Transformed String

The transformed string can be placed in a field described as PIC x(*n*) USAGE BYTE-I18N. This field contains binary data that should be used only for comparisons.

Comparing a Transformed String

If the transformed strings are compared in a program that has PROGRAM COLLATING SEQUENCE in effect, then the string will be transformed again. If this happens, the results of the comparison will be unpredictable.

Buffer Length for Comparing Results

When comparing the results of STRING_TRANSFORM, the buffers being compared should be the same length. If they are not the same length, modify the reference so that the buffer length returned from the service routine can be used. This is possible because *out-buffer* contains binary 0, which fills the buffer to its total length.

Parameters

isl-error-status

is the error structure returned by the service routine.

in-string

is the character variable or literal to be transformed. The maximum length of in-string is 4096 characters.

out-buffer

is the buffer that contains the transformed string returned from the service routine. This buffer must be declared with sufficient size to hold the transformed string. The needed size can be obtained from the TRANSFORM_LENGTH service routine. If the STRING_TRANSFORM operation fails, the contents are undefined.

out-buffer-len

is a one-word binary item defined as PIC 9(10) BINARY, which contains the character length of the transformed string. If the STRING_TRANSFORM operation fails, this length is undefined.

Error Statuses

STRING_TRANSFORM can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
7	7	Encoding Not Supported
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found

Error Code	Severity Code	Description
30	7	Corrupted Table
36	7	String 1 Too Long
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

See 4.4.30 for sample code.

4.4.32. STRPTIME Service Routine

Function Call

UCOB$STRPTM

Description

STRPTIME is used to convert date and time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

General Format

```
CALL 'ucob$strptim' USING status, in-string, ,
      format-string, isl-date-time.
```

Parameters

status

is the error structure returned by the service routine.

in-string

is the string of characters to be converted to integer date-and-time values.

format-string

is the string describing the format of the input string. The format string includes zero or more directives, which consist of one or more white-space characters and either a plain character or a conversion specification.

Directives: The directives are the STRPTIME execution rules. For more details, go to the List of Directives in a Format String.

Plain Character: A plain character is copied unchanged to the output buffer.

Conversion Specification: A conversion specification consists of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

isl-date-time

is a structure where the resulting integer values for the date/time structure are stored. This structure is defined by the ISL-PKT-COPY/COBP PROC.

Return Values

Upon successful completion, the appropriate values are filled into the date and time structure and the procedure returns an error status in *status*.

Error Statuses

STRPTIME can return the following error statuses in the *status* field:

Error Code	Severity Code	Description
0	7	OK (No Error)
3	7	Buffer Too Short
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error
51	7	Invalid Conversion Specification
54	7	Description Does Not Match Input Character
56	7	Invalid Date/Time Value

Sample Code

This code example shows how to use STRPTIME to convert date string characters to values based on locale en_US.8859-1 (English as spoken in the US).

```
           .
           .
           .
 01  string-var  PIC X(20).
 01  format-var  PIC X(12).
*
* Copy in the declaration for isl-error-status.
*
 COPY isl-err-copy.
*
* Copy in the declaration for the isl-date-time structure.
*
 COPY isl-pkt-copy.
           •
           •
           •
     MOVE 'Today is 08/22/95.  ' to string-var.
     MOVE 'Today is %x.' to format-var.
     CALL 'ucob$strptim' using isl-error-status, string-var,
                               format-var, isl-date-time.
*
* Check return status.
*
     IF NOT isl-successful
        IF isl-warning
           DISPLAY 'WARNING' isl-error-num ' RECEIVED'
        ELSE
           DISPLAY 'MAJOR ERROR ' isl-error-num ' RECEIVED'
        END-IF
     ELSE
        DISPLAY 'STRPTIME worked OK'
     END-IF.
     STOP-RUN.
```

Assuming the locale for LC_TIME is "en_US.8859-1", the following values are calculated and placed in these fields within the date and time structure:

```
tm-mday   22

tm-mon 7

tm-year   95
```

4.4.33. TOLOWER Service Routine

Function Call

UCOB$TOLOWER

Description

TOLOWER converts characters in a string to their corresponding lowercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

General Format

```
CALL 'UCOB$TOLOWER' USING isl-error-status, string-in,
          string-out, out-length.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string-in

is the character variable (data name) or literal to be converted. This data name must be defined as a string of one or more characters.

string-out

is the buffer that contains the converted uppercase character string.

out-length

is the character length of *string-out*.

Error Statuses

TOLOWER can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
15	7	Too Many Parameters
16	7	Too Few Parameters
17	7	Overlapping Operands
20	7	Environment Variable Name Contains an Invalid Character

Error Code	Severity Code	Description
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use TOLOWER to convert an input character string to lowercase alphabetic characters.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare variables.
*
 01  strg-in PIC X(5).
 01  strg-out PIC X(10).
 01  out-length PIC 9(10) BINARY.
     .
     .
     .
*
* TOUPPER is called to convert characters in strg-in
* to the corresponding lowercase characters, then
* place them in strg-out.  An error status is returned
* in the variable isl-error-status, and the length of
* the converted characters is returned in the variable
* out-length.
*
     MOVE 'AB12C' TO strg-in.
     CALL 'ucob$tolower' USING isl-error-status, strg-in,
        strg-out, out-length.
*
* If the conversion is successful and if the locale is
* en_US.ASCII, then AB12C and 5 will be displayed.
*
     IF isl-successful
        DISPLAY strg-out, out-length
```

```
ELSE
   DISPLAY 'Conversion has failed'
END-IF.
.
.
.
```

4.4.34. TOUPPER Service Routine

Function Call

UCOB$TOUPPER

Description

TOUPPER converts characters in a string to their corresponding uppercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

General Format

```
CALL 'UCOB$TOUPPER' USING isl-error-status, string-in,
            string-out, out-length.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string-in

is the character variable (data name) or literal to be converted. This data name must be defined as a string of one or more characters.

string-out

is the buffer that contains the converted uppercase character string.

out-length

is the character length of *string-out*.

Error Statuses

TOUPPER can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short

Error Code	Severity Code	Description
15	7	Too Many Parameters
16	7	Too Few Parameters
17	7	Overlapping Operands
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use TOUPPER to convert an input character string to uppercase alphabetic characters.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare variables.
*
 01  strg-in PIC X(5).
 01  strg-out PIC X(10).
 01  out-length PIC 9(10) BINARY.
     .
     .
     .
*
* TOUPPER is called to convert characters in strg-in
* to the corresponding uppercase characters, then
* place them in strg-out.  An error status is returned
* in the variable isl-error-status, and the length of
* the converted characters is returned in the variable
* out-length.
*
    MOVE 'ab12c' TO strg-in.
    CALL 'ucob$toupper' USING isl-error-status, strg-in,
```

```
       strg-out, out-length.
*
* If the conversion is successful and if the locale is
* en_US.ASCII, then AB12C and 5 will be displayed.
*
     IF isl-successful
        DISPLAY strg-out, out-length
     ELSE
        DISPLAY 'Conversion has failed'
     END-IF.
     .
     .
     .
```

4.4.35. TRANSFORM_LENGTH Service Routine

Function Call

UCOB$TRANLEN

Description

TRANSFORM_LENGTH performs these tasks:

- Calculates the maximum length of the ordering key that STRING_TRANSFORM can create for a given string length.
- Determines the maximum size that will be required for the output buffer when using STRING_TRANSFORM. The size that is returned will depend on the current setting of the LC_COLLATE category. It may be several times larger than the value that was passed.

General Format

```
CALL 'UCOB$TRANLEN' USING isl-error-status, string-len,
           transformed-len.
```

Parameters

isl-error-status

is the error structure returned by the service routine.

string-len

is a one-word integer described as PIC 9(10) BINARY, which is the character length of the string to be transformed.

transformed-len

is a one-word integer described as PIC 9(10) BINARY, which is set to the maximum character size required by STRING-TRANSFORM to hold a transformed string of the length specified by *string-len*.

Error Statuses

TRANSFORM_LENGTH can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
15	7	Too Many Parameters
16	7	Too Few Parameters
20	7	Environment Variable Name Contains an Invalid Character
21	7	Environment Variable Value Contains an Invalid Character
26	7	Locale Not Found
30	7	Corrupted Table
39	7	GET$ENV Error
41	7	ASTORE Read Error
42	7	ASTORE Write Error
43	7	Create Bank Error
44	7	Checkpoint Restart Error

Sample Code

This code example shows how to use TRANSFORM_LENGTH to determine the length of the output buffer returned by STRING_TRANSFORM for a specified input string.

```
*
* Copy in the declaration of the return error status.
*
     COPY isl-err-copy.
*
* Declare a variable.
*
 01  strg-var PIC x(6) VALUE 'ABCDEF'.
 01  xfrm-len PIC 9(10) BINARY.
 01  out-area.
     02  out-buf PIC X OCCURS 1 TO 1000 TIMES DEPENDING ON
          xfrm-len.
          .
          .
          .
*
* TRANSFORM_LENGTH is called to determine the length
*  required to transform a string of size 6.
*
```

```
     CALL 'ucob$tranlen' USING isl-error-status, 6, xfrm-len.
     IF NOT isl-successful
       DISPLAY 'An error occurred'
       STOP RUN
     END-IF.
*
* If the size returned is greater than the allocated
*  buffer, then the user must manage the buffer space.
*
     IF xfrm-len > 1000
        DISPLAY 'Output buffer not large enough'
        STOP RUN
     END-IF.
*
* Call STRING-TRANSFORM in order to transform the string.
*
     CALL 'ucob$strxfrm' USING isl-error-status, strg-var,
        out-area, xfrm-len.
          .
          .
          .
```

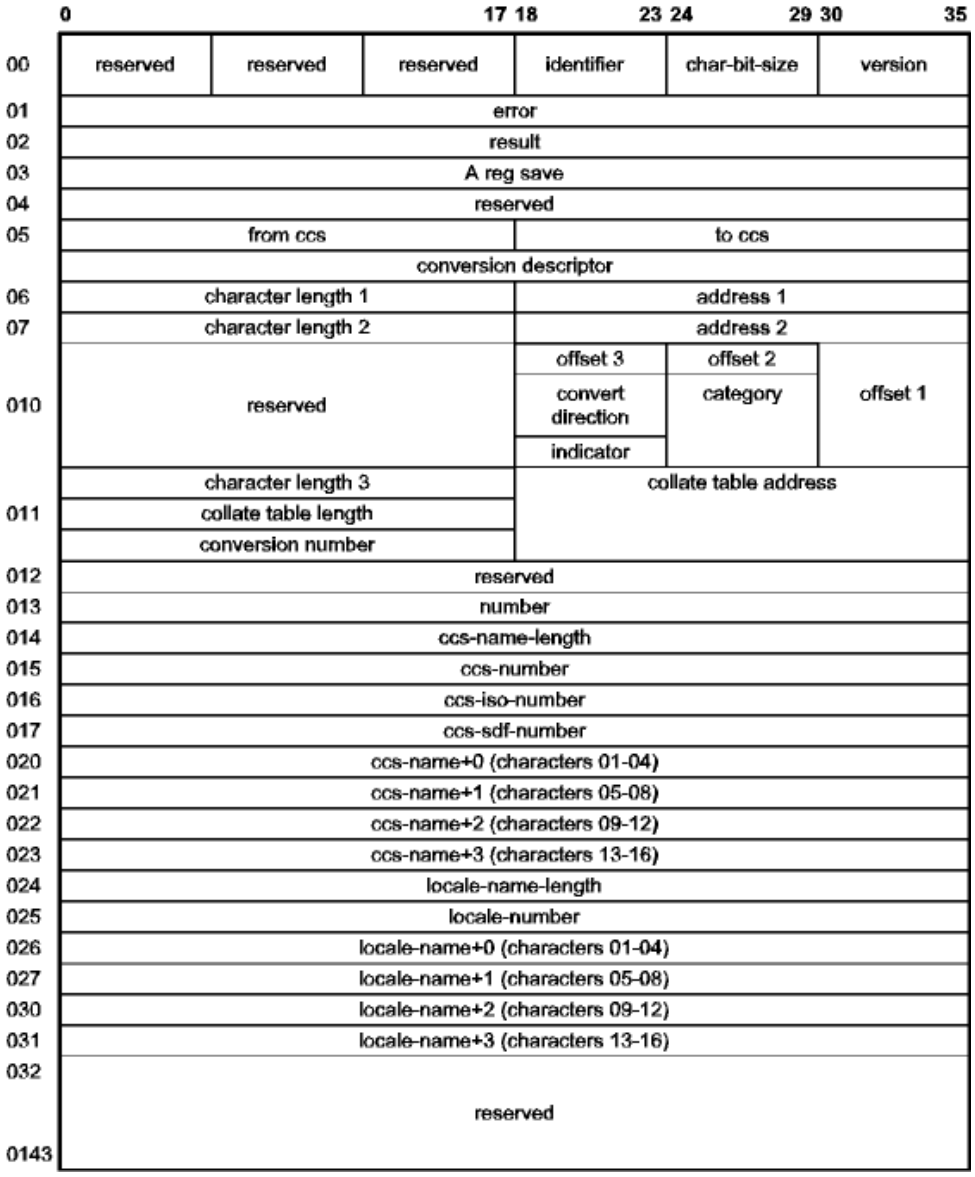
Section 5
Basic Mode MASM

This section describes the I18NLIB service routines available for the Basic Mode MASM language. It also provides information about the INCLUDE elements that define the structures and EQU values used by the service routines.

5.1. Packet for Accessing the I18NLIB Service Routines

The following is the format for the packet that is required by basic mode MASM to call an I18NLIB service routine.

Note: *See the Glossary for definitions and valid contents that apply to all service routines and the Packet Field Settings listed under each service routine to see which ones are required and how they are defined for that service routine.*



5.1.1. Service Routine Packet: Sample Code

This code generates a service routine packet and the related EQUFs.

Note: The code is the same for both ISLPKTGEN and ISLPKTDEFS except for creating the ISLPKT, which applies only to ISLPKTGEN.

```
islpktgen
ISLPKTLEN    $EQU      100            . PACKET LENGTH
                       .
ISLPKT       $RES      ISLPKTLEN      . PACKET
                                      . [APPLIES TO ISLPKTGEN ONLY]
                       .
```

```
slid          $equf     islpkt+0,,s4   . NMNB identifier
islcbitsz     $equf  islpkt+0,,s5      . CCS/LOCALE character bit size
ISLVERSION    $EQUF  ISLPKT+0,,S6      . PACKET VERSION
ISLERROR      $EQUF  ISLPKT+1,,H2      . ERROR CODE RETURN STATUS
ISLERRORWD    $EQUF  ISLPKT+1,,W       . ERROR RETURN STATUS
ISLRESULT     $EQUF  ISLPKT+2,,W       . RESULT FROM FUNCTION
ISLAREGSAVE   $EQUF  ISLPKT+3,,W       . A REGISTER STORE USED BY I18NLIB
                                       .   (INCLUDES WORDS 3 AND 4)
ISLFROMCCS    $EQUF  ISLPKT+5,,H1      . FROM CCS FOR CCS CONVERSION
ISLTOCCS      $EQUF  ISLPKT+5,,H2      . TO CCS FOR CCS CONVERSION
ISLDESCRIPT   $EQUF  ISLPKT+5,,W       . CONVERSION DESCRIPTOR RETURNED BY
                                       .   OPEN_CCS_TO_CCS
ISLCHRLEN1    $EQUF  ISLPKT+6,,H1      . CHARACTER/BYTE LENGTH OF 1ST PARAM
ISLADDR1      $EQUF  ISLPKT+6,,H2      . ADDRESS OF 1ST PARAM
ISLLENADDR1   $EQUF  ISLPKT+6,,W       . LENGTH AND ADDRESS OF 1ST PARAM
ISLCHRLEN2    $EQUF  ISLPKT+7,,H1      . CHARACTER/BYTE LENGTH OF 2ND PARAM
ISLADDR2      $EQUF  ISLPKT+7,,H2      . ADDRESS OF 2ND PARAM
ISLLENADDR2   $EQUF  ISLPKT+7,,W       . LENGTH AND ADDRESS OF 2ND PARAM
ISLOFFSET3    $EQUF  ISLPKT+010,,S4    . CHARACTER/BYTE OFFSET FOR 3RD PARAM
ISLOFFSET2    $EQUF  ISLPKT+010,,S5    . CHARACTER/BYTE OFFSET FOR 2ND PARAM
ISLOFFSET1    $EQUF  ISLPKT+010,,S6    . CHARACTER/BYTE OFFSET FOR 1ST PARAM
ISLCONVDIR    $EQUF  ISLPKT+010,,S4    . DIRECTION TO CONVERT CHARACTER FORM
                                       .   FOR USE WITH CONVERT_FORM
ISLCATEGORY   $EQUF  ISLPKT+010,,S5    . CATEGORY INDICATOR FOR SETLOCALE
ISLINDICATOR  $EQUF  ISLPKT+010,,S4    . USED WITH GET_COLLATE_TABLE
ISLITEM       $EQUF  ISLPKT+010,,S4    . ITEM USED WITH NL_LANGINFO
ISLCHRLEN3    $EQUF  ISLPKT+011,,H1    . CHARACTER/BYTE LENGTH OF 3RD PARAM
ISLCOLADDR    $EQUF  ISLPKT+011,,H2    . COLLATE TABLE ADDRESS
ISLTMADDR     $EQUF  ISLPKT+011,,H2    . START ADDR OF DATE_AND_TIME STRUCT
ISLARGS       $EQUF  ISLPKT+011,,H2    . START ADDR OF ARGUMENTS FOR STRFMON
ISLCOLLEN     $EQUF  ISLPKT+011,,H1    . COLLATE TABLE LENGTH
ISLCNVNUM     $EQUF  ISLPKT+011,,H1    . COLLATE TABLE NUMBER FOR CONVERSIONS
                                       .   BETWEEN NAME AND NUMBER
islnumber     $equf  islpkt+013,,w     . input number
islccsnml     $equf  islpkt+014,,w     . CCS name length
islccsnb      $equf  islpkt+015,,w     . CCS number
islccsisonb   $equf  islpkt+016,,w     . CCS ISO number
islccssdfnb   $equf  islpkt+017,,w     . CCS SDF number
islccsnmw0    $equf  islpkt+020,,w     . CCS name (characters 01-04)
islccsnmw1    $equf  islpkt+021,,w     . CCS name (characters 05-08)
islccsnmw2    $equf  islpkt+022,,w     . CCS name (characters 09-12)
islccsnmw3    $equf  islpkt+023,,w     . CCS name (characters 13-16)
isllocnml     $equf  islpkt+024,,w     . LOCALE name length
isllocnb      $equf  islpkt+025,,w     . LOCALE number
isllocnmw0    $equf  islpkt+026,,w     . LOCALE name (characters 01-04)
isllocnmw1    $equf  islpkt+027,,w     . LOCALE name (characters 05-08)
isllocnmw2    $equf  islpkt+030,,w     . LOCALE name (characters 09-12)
isllocnmw3    $equf  islpkt+031,,w     . LOCALE name (characters 13-16)
```

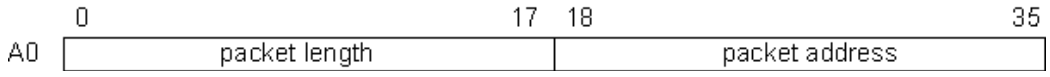
5.1.2. Date/Time Packet: Sample Code

This code generates a date/time packet and the related EQUFs. The code is the same for both ISLTMGEN and ISLTMDEFS, except for line 2 (creating ISLTMPKT), which applies only to ISLTMGEN.

```
ISLTMLEN  $EQU     9                . PACKET LENGTH
.
ISLTMPKT  $RES     ISLTMLEN         . PACKET
.
ISLTMSEC  $EQUF    0                . SECONDS AFTER THE MINUTE [0,
61]
ISLTMMIN  $EQUF    1                . MINUTES AFTER THE HOUR [0, 59]
ISLTMHOUR $EQUF    2                . HOURS SINCE MIDNIGHT [0, 23]
ISLTMMDAY $EQUF    3                . DAY OF THE MONTH [1, 31]
ISLTMMON  $EQUF    4                . MONTHS SINCE JANUARY [0, 11]
ISLTMYEAR $EQUF    5                . YEARS SINCE 1900
ISLTMWDAY $EQUF    6                . DAYS SINCE SUNDAY [0, 6]
ISLTMYDAY $EQUF    7                . DAYS SINCE JANUARY 1 [0,365]
ISLTMDST  $EQUF    8                . DAYLIGHT SAVINGS TIME FLAG
                                    .   -1 = DON'T KNOW
                                    .    0 = OFF DAYLIGHT SAVINGS
                                    .    1 = ON DAYLIGHT SAVINGS
```

5.1.3. Setting Up the Service Routine Packet

Before you can call an I18NLIB service routine, register A0 must contain the packet length in H1 and the packet address in H2. For the current delivery of I18N, the packet length is 100 words.



Not all service routines use all packet fields. Each binding to the service routine indicates which fields are required. If you are a user, you will need to initialize to zero any unused or reserved packet fields.

Some packet fields are overlaid with other field definitions. For a description of the required fields to be completed, refer to each service routine.

5.1.4. Generating and Accessing the Service Routine Packet

The ISLPKTGEN PROC is used to generate the service routine packet for all service routines with bindings to basic or extended mode MASM. This packet is given the externalized label of ISLPKT.

The ISLPKTDEFS PROC provides all the necessary packet EQUFs for accessing ISLPKT. These EQUFs are located in file SYSLIB*PROC$., element ISL-PKT-DEFS/MSM.

Note: This element also contains the variables and values for specific locale item numbers needed by the NL_LANGINFO service routine to determine what locale information to return.

5.1.5. Generating and Accessing the Date/Time Packet

The ISLTMGEN PROC is used to generate the service routine packet for the STRFTIME and STRPTIME service routines for basic and extended mode MASM. This packet is given the externalized label of ISLTMPKT.

The ISLTMDEFS PROC provides all the necessary packet EQUFs for accessing ISLTMPKT. This PROC does not generate the packet itself.

5.1.6. Transferring Control to the Service Routine

Register X11 is used to transfer control to a service routine. When the service routine completes its tasks, it saves and restores all registers except X11.

5.2. Error Structure

Format of Error Status Word

Values

Severity

is a 3-bit integer value with the severity of the error.

0 Successful; the operation succeeded.

2 Informational; the operation did not succeed. The *error-num* field contains more detailed information.

3 Warning; the operation did not succeed. The *error-num* field contains more detailed information.

7 Major error; the operation did not succeed. The *error-num* field contains more detailed information.

Component-id

is a 15-bit unsigned integer with the I18NLIB component-id used by ELMS. This is non-zero even when the service routine executes successfully.

Error-num

is an 18-bit unsigned integer with the error number. EQUs for Error-num values are located in the file SYSLIB*PROC$.ISL-ERR-DEFS/MSM. The value 0 indicates that the service routine succeeded.

5.3. Error Statuses: ISL-ERR-DEFS/MSM

The following equates can be used to interpret the error statuses returned by the basic and extended mode MASM service routines. They can be found in file SYSLIB*PROC$., element ISL-ERR-DEFS/MSM.

$EQU	Service Routine	$EQU	Service Routine
0	ISLOK OK (no error)	32	ISLFROMEQTO From CCS Equals To CCS
1	ISLINVPKTLEN Invalid Packet Length	33	ISLBANKFULL CCS Bank Full
2	ISLINVPKTVER Invalid Packet Version	34	ISLINVBMADDR Invalid Basic Mode Address
3	ISLBUFSHORT Buffer Too Short	35	ISLINVEMENV Invalid Extended Mode Environment
4	ISLFRCCSNDEF From CCS Not Defined	36	ISLSTR12LONG String 1 Too Long
5	ISLTOCCSNDEF To CCS Not Defined	37	ISLSTR22LONG String 2 Too Long
7	ISLENCNOTSUP Encoding Not Supported	38	ISLPUTENVERR PUT$ENV Error
8	ISLENCBUFFAI Encoding of To Buffer Failed	39	ISLGETENVERR GET$ENV Error
13	ISLBADCHAR Bad Character	40	ISLCNUMNOTUN CCS Number Not Unique
14	ISLBADDESC Bad Descriptor	41	ISLASREADERR ASTORE Read Error
15	ISLTOOMANPAR (EMSM only) Too Many Parameters	42	ISLASWRITERR ASTORE Write Error
16	ISLTOOFEWPAR (EMSM only) Too Few Parameters	43	ISLCRTBNKERR Create Bank Error
17	ISLOVERLAPOP Overlapping Operands	44	ISLCPTRSTERR Checkpoint Restart Error
20	ISLNAMINVCHA EV Name Invalid Char	45	ISLCNUMFORUN CCS Number Format Undefined

$EQU	Service Routine	$EQU	Service Routine
21	ISLVALINVCHA EV Value Invalid Char	50	ISLINVLOCITM Invalid Locale Item
22	ISLAREAFULL EV Area Full	51	ISLINVCNVSPC Invalid Conversion Spec
23	ISLNAMILLTIP EV Name Illegal for TIP	52	ISLINVARGMTS Invalid Argument Value
24	ISLNAMNOTFOU EV Name Not Found	53	ISLTOOFEWARG Too Few Arguments
25	ISLVALTOOLON EV Value Too Long	54	ISLCHRNOTMCH Descriptor and Input Chr Don't Match
26	ISLLOCNOTFOU Locale Not Found	55	ISLUNSPMODCS Modified Conversion Spec Not Supported
27	ISLNOTINTER Not a Unisys Intermediate CCS	56	ISLINVDATTIM Invalid Date/Time Value
28	ISLINVCAT Invalid Category	57	ISLINVNMNBID Invalid NMNB Packet Identifier
29	ISLCATNOTSET Category Not Set	58	ISLINVNMLEN Invalid Name Length in NMNB
30	ISLCORRUPTAB Corrupted Table	59	ISLSYSTIMERR SYSTEM$TIME Error
31	ISLCCSNOTFOU CCS Not Found		

5.4. Date/Time Structures (STRFTIME and STRPTIME)

The date/time structure for these two service routines is a nine-word integer packet.

Table 5–1. STRFTIME and STRPTIME Date/Time Structures

This word	Contains or indicates	And must be within this range
0	Number of seconds after the minute. The upper limit is 60 for handling "leap seconds."	[0, 60]
1	Number of minutes after the hour.	[0, 59]
2	Number of hours since midnight.	[0, 23]
3	Day of the month.	[1, 31]
4	Number of months since January. If the current month is January, this number is 0.	[0, 11]

Table 5–1. STRFTIME and STRPTIME Date/Time Structures

This word	Contains or indicates	And must be within this range
5	Number of years since 1900 A -1 in this field indicates the year 1899. A 100 in this word indicates the year 2000.	[-1899, 8099]
6	Number of days since Sunday. If the current day is Sunday, this number is 0.	[0, 6]
7	Number of days since January 1. If the current day is January 1, this number is 0.	[0,365]
8	Status of Daylight Savings Time positive, DST in effect; 0, DST not in effect, and negative, no DST information available	language-specific

5.5. CCS_NAME_TO_CCS_NUM Service Routine

Entry Point

BMSM$CNM2CNB

Description

CCS_NAME_TO_CCS_NUM transforms a coded character set (CCS) name to its corresponding CCS number.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$CNM2CNB
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the string containing the CCS name. This name is case-sensitive and must correspond to a CCS_NAME entity in the Repository for ClearPath OS 2200 and a CCS that is active on the system.

offset 1

is the character position in the string indicated by *address 1*. This is where the CCS name is located.

0 = the name begins on the word boundary (Q1) indicated by *address 1*.

1 = the name begins in Q2.

2 = the name begins in Q3.

3 = the name begins in Q4.

character length 1

is the number of characters in the CCS name.

indicator

is the kind of CCS number desired.

0 = the Unisys CCS number for use with CCS_TO_CCS transliteration.

1 = the ISO CCS number for interoperability use.

2 = the system data format (SDF) number for use as an SDF character set type in SDF control records and data.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is the CCS number corresponding to the specified CCS name. If the service routine returns an error status other than OK, this value is undefined.

Error Statuses

CCS_NAME_TO_CCS_NUM can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 31 – ISLCCSNOTFOU
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV

- 45 – ISLCNUMFORUN

Sample Code

This code example shows how to use CCS_NAME_TO_CCS_NUM to transform a CCS name to its corresponding CCS number.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii                      .
ccsnam    'ISO8859-1'                 . ccs name
ccsnamlen $equ      $sl('ISO8859-1')  . char. length of ccs name
          islpktgen                   . generate islpkt, inc. EQUFs
.
. Messages issued by the program:
.
errmsg    'ERROR STATUS FROM BMSM$CNM2CNB' . print error msg
errmsgl   $equ      $-errmsg          . message length
zeroes    +         0                 . word of zeroes
$(1)                                  .
cnm2cnbex*                            .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . get pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          la,u      a0,ccsnam         . get addr. of ccs name
          lxi,u     a0,ccsnamlen      . get char. len. of ccs name
          sa        a0,isllenaddr1    . save length and address
          la,u      a0,0              . get 1st char. position
          sa        a0,isloffset1     . save name of char. position
          sz        islindicator      . want ccs number?
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$cnm2cnb  . give control to cnm2cnb
          tnz       islerror          . was there an error?
          j         noerror           . no, continue processing
          a$print   (pf 1,errmsgl,errmsg) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               . process returned number
.
. Insert normal processing here
.
rtn1                                  .
          er        exit$             .
          $end      cnm2cnbex         . end cnm2cnb
```

5.6. CCS_NUM_TO_CCS_NAME Service Routine

Entry Point

BMSM$CNB2CNM

Description

CCS_NUM_TO_CCS_NAME transforms a coded character set (CCS) number to its corresponding CCS name.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$CNB2CNM
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

conversion number

is the CCS number. The CCS must participate in a CCS-TO-CCS conversion that is active on the system.

indicator

is the kind of CCS number provided.

0 = is the Unisys CCS number for use with CCS-TO-CCS transliteration.

1 = is the ISO CCS number for interoperability use.

2 = is the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records.

address 1

is a pointer to the string into which the service routine should write the corresponding CCS name. This name is case-sensitive and corresponds to a CCS_NAME entity in the Repository for ClearPath OS 2200. If the service routine returns an error status other than OK, the value for the CCS name is undefined.

offset 1

is the character position in the string indicated by *address 1*. This is where the CCS name is to begin.

0 = the name should begin on the word boundary (Q1) indicated by *address 1*.

1 = the name should begin in Q2.

2 = the name should begin in Q3.

3 = the name should begin in Q4.

character length 1

is the maximum bit length for the CCS name area indicated by *address 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is the bit length of the CCS name. If the service routine returns an error status other than OK, this value is undefined.

address 1

is a pointer to a buffer that contains the CCS name. This value is set by the CCS_NUM_TO_CCS_NAME operation.

If the operation is successful, the buffer contains the name of the CCS corresponding to the conversion number value.

If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 1* string is undefined.

Error Statuses

CCS_NUM_TO_CCS_NAME can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 31 – ISLCCSNOTFOU
- 34 – ISLINVBMADDR

- 35 – ISLINVEMENV
- 40 – ISLCNUMNOTUN
- 45 – ISLCNUMFORUN

Sample Code

This code example shows how to use CCS_NUM_TO_CCS_NAME to transform a CCS number to its corresponding CCS name.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii                      .
ccsnam    $res      4                 . ccs name
ccsnum    +         35                . ccs number
          islpktgen                   . generate islpkt, inc. EQUFs
.
. Messages issued by the program:
.
errmsg    'ERROR STATUS FROM BMSM$CNB2CNM' . print error msg
errmsgl   $equ      $-errmsg          . message length
zeroes    +         0                 . word of zeroes
.
$(1)                                  .
cnb2cnmex*                            .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . get pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          lxi,u     a0,144            . ccs name area bit length
          lxm,u     a0,ccsnam         . get address of ccs name
          sa        a0,isllenaddr1    . save length and address
          la,u      a0,0              . get 1st char. position
          sa        a0,isloffset1     . save char. position
          la        a0,ccsnum         . load ccs number to find
          sa        a0,islcnvnum      . store ccs number to be found
          sz        islindicator      . have a ccs number
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$cnb2cnm  . give control to cnb2cnm
          tnz       islerror          . was there an error?
          j         noerror           . no, continue processing
          a$print   (pf 1,errmsgl,errmsg) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               . process returned ccs name
.
```

```
.  Insert normal processing here
.
rtn1                                    .
          er        exit$               .
          $end      cnb2cnmex           . end cnb2cnm
```

5.7. CCS_TO_CCS Service Routine

Entry Points

BMSM$CCS2CCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$CCS2CCS
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

conversion descriptor

is the internal identifier for the transliteration tables. This identifier is returned by OPEN_CCS_TO_CCS.

address 1

is a pointer to the string containing the characters to be transliterated.

offset 1

is a character position in the string indicated by *address 1*. This is the location of the first character to be transliterated.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 1*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 1

is the number of bytes to be converted, starting at the *offset 1* field.

If the *from_ccs* field in the OPEN_CCS_TO_CCS call is Fieldata, put the number of Fieldata characters to be converted in the *character length 1* field.

address 2

is a pointer to the buffer in which CCS_TO_CCS places the transliterated string.

offset 2

is a character position in the buffer indicated by *address 2*. This is where the transliterated string is to begin.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 2*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 2

is the maximum number of bytes available in the buffer indicated by *address 2*. This is where the transliterated string is to be stored.

If the *to_ccs* field in the OPEN_CCS_TO_CCS call is Fieldata, put in the *character length 2* field the number of Fieldata character spaces available in the buffer indicated by *address 2*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

address 2

is a pointer to a buffer that contains the transliterated characters. This value is set by the CCS_TO_CCS operation.

If the operation is successful, the area indicated by *address 2* contains the transliterated characters from the string indicated by *address 1*.

If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 2* string is undefined.

result

is the bit length of the transliterated string indicated by *address 2*.

Sample Code

See 5.8 for sample code.

5.8. CLOSE_CCS_TO_CCS Service Routine

Entry Points

BMSM$CLOSCCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$CLOSCCS
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

conversion descriptor

is the internal identifier for the transliteration tables. This identifier is returned by the OPEN_CCS_TO_CCS service routine.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

Sample Code

This code example shows how to use OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS to transliterate a character string from one CCS to another.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
from_ccs  $equ      19                . ISO 646FR in from_ccs
                                      . field
to_ccs    $equ      35                . ISO 8859.1 in to_ccs
                                      . field
cv_des    $res      1                 . conversion descriptor
f_string  'résumé'                    . define from_string
fstrlen   $sl('résumé')               . char. length of string
outbuf    $res      2                 . output buffer
outlen    $equ      8                 . output length in char.
zeroes    +         0                 . word of zeroes
.
msg1      'ERROR STATUS FROM BMSM$OPENCCS' . print error msg
msg1l     $equ      $-msg1            . message length
.
msg2      'OPEN_CCS_TO_CCS WORKED'    . msg indicates no error
msg2l     $equ      $-msg2            . message length
.
msg3      'ERROR STATUS FROM BMSM$CCS2CCS'. print error msg
msg3l     $equ      $-msg3            . message length
.
msg4      'CCS_TO_CCS WORKED'         . msg indicates no error
msg4l     $equ      $-msg4            . message length
.
          islpktgen                   . generate islpkt, get EQUFs
.
$(1)                                  .
ccstoccsex*                           .
          la        a0,(0,zeroes)     . get source for BT
          la        a1,(1,islpkt)     . get destination for BT
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize entire packet
          la,u      a0,from_ccs       . get from_ccs number
          sa        a0,islfromccs     . save the from_ccs
          la,u      a0,to_ccs         . get to_ccs number
          sa        a0,isltoccs       . save to_ccs
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$openccs  . give control to openccs
          tnz       islerror          . was there an error?
.
. OPEN_CCS_TO_CCS was successful.  The error status
. and conversion descriptor have been saved in the packet.
.
          j         nextcall          . prepare for ccs2ccs call
          a$print   (pf 1,msg1l,msg1) . print error msg
```

```
          j         rtn1              . jump to common code
nextcall                              .
          a$print   (pf 1,msg2l,msg2) . print no error msg
          la        a1,islresult      . get result of openccs
          sa        a1,cv_des         . save descriptor in cv_des
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     .
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          la        a0,cv_des         . get conversion descriptor
          sa        a0,isldescript    . save conversion desc.
          la,u      a0,f_string       . get addr. of from_string
          lxi       a0,fstrlen        . get length of f_string
          sa        a0,isllenaddr1    . save from-ccs info
          la,u      a0,0              . set byte offset
          sa        a0,isloffset1     . save from_offset
          la,u      a0,outbuf         . get output buffer addr.
          lxi,u     a0,outlen         . get maximum length
          sa        a0,isllenaddr     . save information
          la,u      a0,0              . set byte offset of outbuf
          sa        a0,isloffset2     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$ccs2ccs  . give control to ccs2ccs
          tnz       islerror          . was there an error?
.
. CCS_TO_CCS was successful.  The output buffer contains the
. converted string.  The bit length of the converted string has
. been saved in the result word of the BMSM service routine
. packet, and any error status has been placed in the error
. word of the packet.
.
          j         closecall         . ready for closccs
          a$print   (pf 1,msg3l,msg3) . print error msg
          j         rtn1              . jump to common code
closecall                             .
          a$print   (pf 1,msg4l,msg4) . print no error msg
          la        a0,(0,zeroes)     . prepare for initialization
          la        a1,(1,islpkt)     . get pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . clear packet
          la        a0,cv_des         . load descriptor into A0
          sa        a0,isldescript    . save conversion descriptor
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$closccs  . give control to closccs
rtn1                                  .
          er        exit$             .
          $end      ccstoccsex        . end string transliterate
```

5.9. CONVERT_FORM Service Routine

Entry Point

BMSM$CNVRTFM

Description

CONVERT_FORM performs these tasks:

- Converts a character from its current representation in one portion of a character set to its representation in another portion of the same character set. There is always a one-to-one correspondence in the transliteration that occurs.
- Uses the current value of the locale in the LC_CTYPE category to determine which transliteration table to use. The calling program must ensure that LC_CTYPE is set to a locale that contains the appropriate transliteration table.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$CNVRTFM
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the string containing the characters to be converted.

offset 1

is a character position in the string indicated by *address 1*. This is the location of the first character to convert.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 1*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 1

is the number of bytes to be converted, starting at *offset 1*.

address 2

is a pointer to the buffer in which the service routine places the converted character string.

The output buffer can be the same as the input buffer; it must be at least as large as the input string.

offset 2

is a character position in the buffer indicated by *address 2*. This is where the converted string is to begin.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 2*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

convert direction

indicates if the characters are to be converted to the 7-bit or 8-bit range.

0 = convert to the 7-bit range.

1 = convert to the 8-bit range.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

address 2

is a pointer to a buffer that contains the transliterated characters. This value is set by the CONVERT_FORM operation.

- If the operation is successful, the area indicated by *address 2* contains the transliterated characters from the string indicated by *address 1*.
- If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 2* string is undefined.

Error Statuses

CONVERT_FORM can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN

- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 17 – ISLOVERLAPOP
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 27 – ISLNOTINTER
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code routine shows how to use CONVERT_FORM to change the encoding of a character string.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
a_string  'süß'                       . define a string
strlen    $sl('süß')                  . char. length of string
outbuf    $res      5                 . convert_form output buffer
.
msg1      'ERROR STATUS FROM BMSM$CNVRTFM' . print error msg
msg1l     $equ      $-msg1            . message length
.
msg2      'CONVERT_FORM WORKED'       . message for no error
msg2l     $equ      $-msg2            . message length
.
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt, inc. EQUFs
.
$(1)                                  .
cnvrtfmex*                            .
          la        a0,(0,zeroes)     . prepare for initialization
          la        a1,(1,islpkt)     . get pkt. addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . clear packet
```

```
          la,u      a0,a_string       . get address of string
          lxi       a0,strlen         . get character length of
                                      .    string to transliterate
          sa        a0,isllenaddr1    . save length and address
          la,u      a0,0              . get byte offset
          sa        a0,isloffset1     . save byte offset
          la,u      a0,outbuf         . get outbuf address
          lxi,u     a0,5*4            . buffer character length
          sa        a0,isllenaddr2    . save outbuf address
          la,u      a0,0              . set byte offset for outbuf
          sa        a0,isloffset2     . save buffer offset in pkt
          la,u      a0,1              . trans. fr 7- to 8-bit range
          sa        a0,islindicator   . save translit.direction
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$cnvrtfm  . give control to cnvrtfm
          tnz       islerror          . was there an error?
.
. CONVERT_FORM was successful; the out buffer contains the
. transliterated string.
.
          j         noerror           . no, continue processing
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1                . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      cnvrtfmex         . end cnvrtfm
```

5.10. GET_COLLATE_TABLE Service Routine

Entry Point

BMSM$GCOLTBL

Description

GET_COLLATE_TABLE is a MASM-callable service routine that copies the collation tables from the I18NLIB common bank into the application's memory space. These tables are indicated by the current setting of the LC_COLLATE category. This operation

- Allows faster processing for the STRING_COLLATE and STRING_TRANSFORM service routines.
- Enables these service routines to access the collating rules directly from the application data space rather than having to establish a pointer to the I18NLIB AFCB.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$GCOLTBL
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the buffer containing the name of the locale.

offset 1

is a character position in the string indicated by *address 1*. This is where the locale name is located.

0 = the name should begin on the word boundary (Q1) indicated by *address 1*.

1 = the name should begin in Q2.

2 = the name should begin in Q3.

3 = the name should begin in Q4.

character length 1

is the character length of the locale name indicated by *address 1*. The *result* setting depends on *character length 1*.

- If *character length 1* is 0, GET_COLLATE_TABLE uses the locale name indicated by the LC_COLLATE category.
- If *character length 1* is not 0, GET_COLLATE_TABLE uses the locale name indicated by *address 1*.

indicator

is an integer.

- If *indicator* is 0, GET_COLLATE_TABLE returns the length of the collate table in the *result* field.
- If *indicator* is 1, GET_COLLATE_TABLE copies the collate table to the area indicated by *collate table address*.

collate table address

is the Dbank relative address of the location where the collate table is to be copied.

- If *character length 1* is 0, the collate table copied is the one indicated by the LC_COLLATE category.

- If *character length 1* is not 0, the collate table copied is the one indicated by *address 1*.

collate table length

is the number of words available at *collate table address*, which is where the collate table is to be copied. Any unused portion of the space provided remains unaltered.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

collate table address

is the data Dbank relative address of the location where the collate table is to be copied. This area is set by the GET_COLLATE_TABLE operation.

- If the operation is successful, the area indicated by *address 2* contains the collate table from the locale indicated by *address 1* or indicated by the LC_COLLATE category.
- If the operation is not successful, an error status is returned that indicates why it failed, and the contents of the *address 2* data area are undefined.

result

- If the indicator field of the packet on input to the routine is 0, result contains the number of words required to save the collate table for the locale indicated either by *address 1* or by the LC_COLLATE category.
- If the indicator field is 1, result contains the number of words transferred to the buffer area.
- If a Buffer Too Short error status is returned, the result field contains the number of words required to save the collate table for the locale indicated either by *address 1* or by the LC_COLLATE category.

Error Statuses

GET_COLLATE_TABLE can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB

- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42-ISLASTRWRITE
- 43-ISLCREATBANK
- 44-ISLCHPTRESRT

Sample Code

This code routine shows how to use GET_COLLATE_TABLE to copy the collation table for the locale indicated by LC_COLLATE into the buffer col_table.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
col_size  $equ      500               . 500 words for lc_collate
                                      . tbl
col_table  $res     col_size          . reserve space
msg1      'ERROR STATUS FROM BMSM$GCOLTBL' . message for error
msg1l     $equ      $-msg1            . message length
msg2      'COLLATE TABLE RETRIEVED'   . message for no error
msg2l     $equ      $-msg2            . message length
          islpktgen                   . generate islpkt, inc.
                                      . EQUFs
collenadd  $equf    islpkt+8,,w       . collate tbl len. & addr.
.
$(1)                                  .
getcolex*                             .
          la        a0,(0,zeroes)     .
          la        a1,(1,islpkt)     . load pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          la,u      a0,col_size       . get # of words available
          sa        a0,islcollen      . save into packet
          la,u      a0,col_table      . get addr. for saving tbl
          sa        a0,islcoladdr     . save into packet
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
.
. Call service routine to get LC_COLLATE table for locale in force.
.
          i$bj      x11,bmsm$gcoltbl  . give control to gcoltbl
          tnz       islerror          . was there an error?
          j         noerror           . no
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
```

```
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      getcolex          . end get_collate_table
```

5.11. GETENV Service Routine

Entry Point

BMSM$GETENV

Description

GETENV retrieves the current value of the environment variable set at the run level. Run-level environment variables are initially set by the default values from the user profile.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$GETENV
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to a string that contains the environment variable name.

character length 1

is the length, in characters, of the string pointed to by *address 1*.

address 2

is a pointer to a buffer in which the service routine writes the value of the environment variable indicated by *address 1*.

character length 2

is the length, in characters, of the area indicated by *address 2*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

address 2

is a pointer to a string that contains the value of the environment variable. This value is set by the service routine operation.

If the operation is successful, the area indicated by *address 2* contains the value of the environment variable indicated by *address 1*.

If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 2* string is undefined.

result

is the length, in characters, of the environment variable value. If the operation fails, the value of result is undefined.

Error Statuses

GETENV and GETENVSYS can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
20	7	EV Name Contains an Invalid Character
21	7	EV Value Contains an Invalid Character
24	3	EV Name Not Found
39	7	GET$ENV Error

Sample Code

This code example shows how to use GETENV to retrieve the value of environment variables (EV) at the run level.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
```

```
cpw       $equ      4                 . 4 ASCII char. per word
.
. Environment variable name character size definitions:
.
lc_col_sz  $equ     $sl('LC_COLLATE') . char. length of
                                      .   LC_COLLATE
lc_ctyp_sz  $equ    $sl('LC_CTYPE')   . char. length of LC_CTYPE
lc_all_sz  $equ     $sl('LC_ALL')     . char. length of LC_ALL
lang_sz   $equ      $sl('LANG')       . char. length of LANG
ev_max_word  $equ   4                 . max. # words for EV name
ev_val_sz  $equ     ev_max_word*cpw   . max. # char. for EV name
.
. Environment variable name definitions:
.
lc_collate  'LC_COLLATE'              . LC_COLLATE EV name
lc_ctype  'LC_CTYPE'                  . LC_CTYPE EV name
lc_all    'LC_ALL'                    . LC_ALL EV name
lang      'LANG'                      . LANG EV name
ev_value  $res      ev_max_word       . storage area for EV value
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt, get EQUFs
msg2      'ERROR STATUS FROM BMSM$GETENV' . print error msg
msg2l     $equ      $-msg2            . message length
msg3      'GETENV WORKED'             . message for no error
msg3l     $equ      $-msg3            . message length
.
$(1)                                  .
getenvex*                             .
          la        a0,(0,zeroes)     . get source for BT
          la        a1,(1,islpkt)     . get destination for BT
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire pkt
          la,u      0,lc_collate      . get addr. of EV name
          lxi,u     a0,lc_col_sz      . get char. size of EV name
          sa        a0,isllenaddr1    . save EV name length & addr
          la,u      a0,ev_value       . get EV value save area addr
          lxi,u     a0,ev_val_sz      . get # char. of EV value
          sa        a0,isllenaddr2    . save length and addr.
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$getenv   . give control to getenv
          tnz       islerror          . was there an error?
          j         noerror           . no, continue processing
          a$print   (pf 1,msg2l,msg2) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg3l,msg3) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      getenvex          . end getenv
```

5.12. GETENVSYS Service Routine

Entry Point

BMSM$GETENVS

Description

GETENVSYS retrieves the current value of the environment variable set at the system level.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$GETENVS
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to a string that contains the environment variable name.

character length 1

is the length, in characters, of the string pointed to by *address 1*.

address 2

is a pointer to a buffer in which the service routine writes the value of the environment variable indicated by *address 1*.

character length 2

is the length, in characters, of the area indicated by *address 2*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

address 2

is a pointer to a string that contains the value of the environment variable. This value is set by the service routine operation.

If the operation is successful, the area indicated by *address 2* contains the value of the environment variable indicated by *address 1*.

If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 2* string is undefined.

result

is the length, in characters, of the environment variable value. If the operation fails, the value of result is undefined.

Error Statuses

GETENV and GETENVSYS can return the following error statuses in the *isl-error-num* record:

Error Code	Severity Code	Description
0	0	OK (No Error)
3	7	Buffer Too Short
20	7	EV Name Contains an Invalid Character
21	7	EV Value Contains an Invalid Character
24	3	EV Name Not Found
39	7	GET$ENV Error

Sample Code

This code example shows how to use GETENVSYS to retrieve the value of environment variables (EV) at the system level.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
cpw       $equ      4                 . 4 ASCII char. per word
.
. Environment variable name character size definitions:
.
lc_col_sz  $equ     $sl('LC_COLLATE') . char. length of
                                      .   LC_COLLATE
lc_ctyp_sz  $equ    $sl('LC_CTYPE')   . char. length of LC_CTYPE
```

```
lc_all_sz  $equ     $sl('LC_ALL')     . char. length of LC_ALL
lang_sz   $equ      $sl('LANG')       . char. length of LANG
ev_max_word  $equ   4                 . max. # words for EV name
ev_val_sz  $equ     ev_max_word*cpw   . max. # char. for EV name
.
. Environment variable name definitions:
.
lc_collate  'LC_COLLATE'              . LC_COLLATE EV name
lc_ctype  'LC_CTYPE'                  . LC_CTYPE EV name
lc_all    'LC_ALL'                    . LC_ALL EV name
lang      'LANG'                      . LANG EV name
ev_value  $res      ev_max_word       . storage area for EV value
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt, get EQUFs
msg2      'ERROR STATUS FROM BMSM$GETENVS' . print error msg
msg2l     $equ      $-msg2            . message length
msg3      'GETENVS WORKED'            . message for no error
msg3l     $equ      $-msg3            . message length
.
$(1)                                  .
getenvsyex*                           .
          la        a0,(0,zeroes)     . get source for BT
          la        a1,(1,islpkt)     . get destination for BT
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a0,lc_collate     . get addr. of EV name
          lxi,u     a0,lc_col_sz      . get char. size of EV name
          sa        a0,isllenaddr1    . save len. & addr. of
                                      . EV name
          la,u      a0,ev_value       . get EV value save area
                                      . addr.
          lxi,u     a0,ev_val_sz      . get # char. of EV value
          sa        a0,isllenaddr2    . save length and addr.
          la,u      a0,islpkt         . get packet address
          lxi,u a0,islpktlen          . get packet length
          i$bj      x11,bmsm$getenvs  . give control to getenvsys
          tnz       islerror          . was there an error?
          j         noerror           . no, continue processing
          a$print   (pf 1,msg2l,msg2) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg3l,msg3) . print no error msg
rtn1                                  .
          er        exit$             .
          $end getenvsex              . end getenvsys
```

5.13. ISALNUM Service Routine

Entry Point

BMSM$ISALNUM

Description

ISALNUM determines if all the characters in the string are in the alphanumeric character class. If they are, it returns TRUE.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *alpha* and *digit* categories in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISALNUM
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISALNUM to determine if all characters of the input string are alphanumeric.

```
           $include  'maxr$'           .
           $include  'eru$'            .
           $include  'isl-pkt-defs/msm' .
pf         $form     12,6,18           .
$(0)                                   .
           $ascii                      .
```

```
pf        $form     12,6,18           .
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISALNUM' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISALNUM WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
isalnumex*                            .
          la        a0,(0,zeroes)     . get addr.of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$isalnum  . give control to isalnum
          tz        islresult         . all characters
                                      . alphanumeric?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not alphanumeric
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      isalnumex         . end isalnum
```

5.14. ISALPHA Service Routine

Entry Point

BMSM$ISALPHA

Description

ISALPHA determines if all the characters in the string are in the alphabetic character class. If they are, it returns TRUE.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *upper*, *lower*, and *letter* categories in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISALPHA
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISALPHA to determine if all characters of the input string are alphabetic.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii                      .
pf        $form     12,6,18           .
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISALPHA' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISALPHA WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
isalphaex*                             .
          la        a0,(0,zeroes)     . get addr.of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
```

```
            lxi,u     a1,strlen         . get character length
            sa        a1,isllenaddr1    . save length and address
            la,u      a2,0              . get byte offset of string
            sa        a2,isloffset1     . save byte offset
            la,u      a0,islpkt         . get packet address
            lxi,u     a0,islpktlen      . get packet length
            i$bj      x11,bmsm$isalpha  . give control to isalpha
            tz        islresult         . all characters alphabetic?
            j         noerror           . yes
            tnz       islerror          . was there an error?
            j         rtn1              . no, item not alphabetic
            a$print   (pf 1,msg1l,msg1) . yes, print error msg
            j         rtn1              . jump to common code
  noerror                               .
            a$print   (pf 1,msg2l,msg2) . print no error msg
  rtn1                                  .
            er        exit$             .
            $end      isalphaex         . end isalpha
```

5.15. ISCNTRL Service Routine

Entry Point

BMSM$ISCNTRL

Description

ISCNTRL determines if all the characters in the string are in the control character class. If they are, it returns TRUE.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *cntrl* category in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISCNTRL
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB

- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISCNTRL to determine if all characters of the input string are control characters.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii                      .
pf        $form     12,6,18
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISCNTRL' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISCNTRL WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
iscntrlex*                            .
          la        a0,(0,zeroes)     . get addr. of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$iscntrl  . give control to iscntrl
          tz        islresult         . all char. control chars.?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not control chars
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
```

```
rtn1                                    .
          er        exit$               .
          $end      iscntrlex           . end iscntrl
```

5.16. ISDIGIT Service Routine

Entry Point

BMSM$ISDIGIT

Description

ISDIGIT determines if all the characters in the string are decimal digits in the numeric character class. If they are, ISDIGIT returns TRUE.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *digit* category in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISDIGIT
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISDIGIT to determine if all the characters of the input string are numeric (decimal digits).

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii                      .
pf        $form     12,6,18           .
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISDIGIT' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISDIGIT WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
isdigitex*                            .
          la        a0,(0,zeroes)     . get addr. of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$isdigit  . give control to isdigit
          tz        islresult         . all characters numeric?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not numeric
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      isdigitex         . end isdigit
```

5.17. ISGRAPH Service Routine

Entry Point

BMSM$ISGRAPH

Description

ISGRAPH determines if all the characters in the string are in the graphic character class. If they are, it returns TRUE.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *graph* category in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISGRAPH
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISGRAPH to determine if all the characters of the input string are graphic characters.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii                      .
pf        $form     12,6,18           .
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISGRAPH' . print error msg
msg1l     $equ      $-msg1            . message length
```

```
msg2      'ISGRAPH WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
isgraphex*                            .
          la        a0,(0,zeroes)     . get addr. of word of zeroes
          la        a1,(1,islpktgen)  . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$isgraph  . give control to isgraph
          tz        islresult         . all characters graphic?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not graphic
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      isgraphex         . end isgraph
```

5.18. ISLOWER Service Routine

Entry Point

BMSM$ISLOWER

Description

ISLOWER determines if all the characters in the string are in the lowercase character class. If they are, it returns TRUE.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *lower* category in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISLOWER
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISLOWER to determine if the characters of the input string are lowercase and alphabetic.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
$(0)                                  .
          $ascii                      .
pf        $form     12,6,18           .
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISLOWER' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISLOWER WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
islowerex*                            .
          la        a0,(0,zeroes)     . get addr. of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
```

```
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$islower  . give control to islower
          tz        islresult         . all characters lowercase?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not lowercase
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      islowerex         . end islower
```

5.19. ISPRINT Service Routine

Entry Point

BMSM$ISPRINT

Description

ISPRINT determines if all the characters in the string are in the printable character class. If they are, it returns TRUE.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *print* category in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISPRINT
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB

- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISPRINT to determine if the characters of the input string are printable.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
$(0)                                  .
          $ascii                      .
pf        $form     12,6,18           .
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISPRINT' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISPRINT WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                   .
isprintex*                            .
          la        a0,(0,zeroes)     . get addr. of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$isprint  . give control to isprint
          tz        islresult         . all characters printable?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not printable
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
```

```
er        exit$             .
$end      isprintex         . end isprint
```

5.20. ISPUNCT Service Routine

Entry Point

BMSM$ISPUNCT

Description

ISPUNCT determines if all the characters in the string are in the punctuation character class. If they are, it returns TRUE.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *punct* category in locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISPUNCT
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISPUNCT to determine if the characters of the input string are punctuation characters.

```
          $include  'maxr$'            .
          $include  'eru$'             .
          $include  'isl-pkt-defs/msm' .
$(0)                                   .
          $ascii                       .
pf        $form     12,6,18            .
str       'stringvalue'                . define a string
strlen    $sl('stringvalue')           . char. length of string
zeroes    +         0                  . word of zeroes
          islptkgen                    . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISPUNCT' . print error msg
msg1l     $equ      $-msg1             . message length
msg2      'ISPUNCT WORKED'             . message for no error
msg2l     $equ      $-msg2             . message length
$(1)                                   .
ispunctex*                             .
          la        a0,(0,zeroes)      . get addr. of word of zeroes
          la        a1,(1,islpkt)      . get islpkt addr. & increment
          lr,u      r1,islpktlen       . get length of packet
          bt        a1,,*a0            . initialize entire packet
          la,u      a1,str             . get length of string
          lxi,u     a1,strlen          . get character length
          sa        a1,isllenaddr1     . save length and address
          la,u      a2,0               . get byte offset of string
          sa        a2,isloffset1      . save byte offset
          la,u      a0,islpkt          . get packet address
          lxi,u     a0,islpktlen       . get packet length
          i$bj      x11,bmsm$ispunct   . give control to ispunct
          tz        islresult          . all characters punctuation?
          j         noerror            . yes
          tnz       islerror           . was there an error?
          j         rtn1               . no, item not punctuation
          a$print   (pf 1,msg1l,msg1)  . yes, print error msg
          j         rtn1               . jump to common code
noerror                                .
          a$print   (pf 1,msg2l,msg2)  . print no error msg
rtn1                                   .
          er        exit$              .
          $end      ispunctex          . end ispunct
```

5.21. ISSPACE Service Routine

Entry Point

BMSM$ISSPACE

Description

ISSPACE determines if all the characters in the string are in the white space character class. If they are, it returns TRUE.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *space* category in locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISSPACE
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISSPACE to determine if the characters of the input string are white space characters.

```
            $include  'maxr$'           .
            $include  'eru$'            .
            $include  'isl-pkt-defs/msm' .
  $(0)                                  .
            $ascii                      .
  pf        $form     12,6,18           .
```

```
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISSPACE' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISSPACE WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
isspaceex*                            .
          la        a0,(0,zeroes)     . get addr. of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$isspace  . give control to isspace
          tz        islresult         . all characters spaces?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not spaces
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      isspaceex         . end isspace
```

5.22. ISUPPER Service Routine

Entry Point

BMSM$ISUPPER

Description

ISUPPER determines if all the characters in the string are in the uppercase character class. If they are, it returns TRUE.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the **upper** category in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISUPPER
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISUPPER to determine if the characters of the input string are uppercase and alphabetic.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
$(0)                                  .
          $ascii                      .
pf        $form     12,6,18           .
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISUPPER' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISUPPER WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
isupperex*                            .
          la        a0,(0,zeroes)     . get addr. of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
```

```
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$isupper  . give control to isupper
          tz        islresult         . all characters uppercase?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not uppercase
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      isupperex         . end isupper
```

5.23. ISXDIGIT Service Routine

Entry Point

BMSM$ISXDIG

Description

ISXDIGIT determines if all the characters in the string are in the hexadecimal digit (xdigit) character class. If they are, it returns TRUE.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *xdigit* category in the locale definition file.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$ISXDIG
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the characters to be tested.

offset 1

is the character position in the string indicated by *address 1*. This is the location of the first character to be tested, that is, the word boundary (Q*n*) indicated by *address 1*.

0 = the test should begin in Q1.

1 = the test should begin in Q2.

2 = the test should begin in Q3.

3 = the test should begin in Q4.

character length 1

the number of characters to be tested, starting at *offset 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If *result* is 0, check *error* to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in the *error* field by the character class service routines.

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB

- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISXDIGIT to determine if the characters of the input string are hexadecimal digits.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
$(0)                                  .
          $ascii                      .
pf        $form     12,6,18           .
str       'stringvalue'               . define a string
strlen    $sl('stringvalue')          . char. length of string
zeroes    +         0                 . word of zeroes
          islptkgen                   . generate islpkt, inc. EQUFs
msg1      'ERROR STATUS FROM BMSM$ISXDIGIT' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'ISXDIGIT WORKED'           . message for no error
msg2l     $equ      $-msg2            . message length
$(1)                                  .
isxdigitex*                           .
          la        a0,(0,zeroes)     . get addr. of word of zeroes
          la        a1,(1,islpkt)     . get islpkt addr. & increment
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a1,str            . get length of string
          lxi,u     a1,strlen         . get character length
          sa        a1,isllenaddr1    . save length and address
          la,u      a2,0              . get byte offset of string
          sa        a2,isloffset1     . save byte offset
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$isxdig   . give control to isxdigit
          tz        islresult         . all characters hex digits?
          j         noerror           . yes
          tnz       islerror          . was there an error?
          j         rtn1              . no, item not hex digit
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                  .
```

```
er        exit$             .
$end      isxdigitex        . end isxdigit
```

5.24. LOCALE_NAME_TO_CCS_NUM Service Routine

Entry Point

BMSM$LNM2CNB

Description

LOCALE_NAME_TO_CCS_NUM returns the coded character set (CCS) number associated with the locale name.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$LNM2CNB
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the string containing the locale name. This name is case-sensitive and must correspond to a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and a locale that is active on the system.

offset 1

is the character position in the string indicated by *address 1*. This is where the locale name is located.

0 = the name begins on the word boundary (Q1) indicated by *address 1*.

1 = the name begins in Q2.

2 = the name begins in Q3.

3=the name begins in Q4.

character length 1

is the number of characters in the locale name.

indicator

is the kind of CCS number desired:

0 = is the Unisys CCS number for use with CCS_TO_CCS transliteration.

1 = is the ISO CCS number for interoperability use.

2 = is the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is the CCS number corresponding to the specified name. If the service routine returns an error status other than OK, this value is undefined.

Error Statuses

LOCALE_NAME_TO_CCS_NUM can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 26 – ISLLOCNOTFOU
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 45 – ISLCNUMFORUN

Sample Code

This code example shows how to use LOCALE_NAME_TO_CCS_NUM to return the CCS number associated with the locale name.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
locnam    'fr_FR.8859-1'              . locale name
locnamlen $equ      $sl('fr_FR.8859-1') . char. length of locale name
          islpktgen                   . generate islpkt, inc. EQUFs
.
```

```
. Messages issued by the program:
.
errmsg    'ERROR STATUS FROM BMSM$LNM2CNB' . print error msg
errmsgl   $equ      $-errmsg          . message length
zeroes    +         0                 . word of zeroes
.
$(1)                                  .
lnm2cnbex*                            .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . get pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          la,u      a0,locnam         . get addr. of locale name
          lxi,u     a0,locnamlen      . get char. length
          sa        a0,isllenaddr1    . save length and address
          la,u      a0,0              . get 1st char. position
          sa        a0,isloffset1     . save name of char position
          sz        islindicator      . want the ccs number?
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$lnm2cnb  . give control to lnm2cnb
          tnz       islerror          . was there an error?
          j         noerror           . no, continue proc. message
          a$print   (pf1,errmsg1,errmsg) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
.
. Process the returned number.
.
rtn1                                  .
          er        exit$             .
          $end      lnm2cnbex         . end lnm2cnb
```

5.25. LOCALE_NAME_TO_LOCALE_NUM Service Routine

Entry Point

BMSM$LNM2LNB

Description

LOCALE_NAME_TO_LOCALE_NUM transforms a locale name to its corresponding locale number.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$LNM2LNB
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the string containing the locale name. This name is required by the SETLOCALE service routine. It is case-sensitive and must correspond to a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and a locale that is active on the system.

offset 1

is the character position in the string indicated by *address 1*. This is where the locale name is located.

0 = the name begins on the word boundary (Q1) indicated by *address 1*.

1 = the name begins in Q2.

2 = the name begins in Q3.

3 = the name begins in Q4.

character length 1

is the number of characters in the locale name.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is the locale number corresponding to the specified name If the service routine returns an error status other than OK, this value is undefined.

Error Statuses

LOCALE_NAME_TO_LOCALE_NUM can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN

- 2 – ISLINVPKTVER
- 26 – ISLLOCNOTFOU
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV

Sample Code

This code example shows how to use LOCALE_NAME_TO_LOCALE_NUM to transform a locale name to its corresponding locale number.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii                      .
locnam    'fr_FR.8859-1'              . locale name
locnamlen $equ      $sl('fr_FR.8859-1') . char. len. of locname
          islpktgen                   . generate islptk, inc. EQUFs
.
. Messages issued by the program:
.
errmsg    'ERROR STATUS FROM BMSM$LNM2LNB' . print error msg
errmsgl   $equ      $-errmsg          . message length
zeroes    +         0                 . word of zeroes
.
$(1)                                  .
lnm2lnbex*                            .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . get pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          la,u      a0,locnam         . get addr. of locale name
          lxi,u     a0,locnamlen      . get char. len. of locname
          sa        a0,isllenaddr1    . save length and address
          la,u      a0,0              . get 1st char. position
          sa        a0,isloffset1     . save name of char. position
          la,u      a0,islpkt         . get address of packet
          lxi,u     ao,islpktlen      . get packet length
          i$bj      x11,bmsm$lnm2lnb  . give control to lnm2lnb
          tnz       islerror          . was there an error?
          j         noerror           . no, continue processing
          a$print   (pf 1,errmsgl,errmsg) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
.
. Process the returned number.
.
rtn1                                  .
          er        exit$             .
          $end      lnm2lnbex         . end lnm2lnb
```

5.26. LOCALE_NUM_TO_CCS_NUM Service Routine

Entry Point

BMSM$LNB2CNB

Description

LOCALE_NUM_TO_CCS_NUM returns the coded character set (CCS) number associated with the locale number.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$LNB2CNB
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

conversion number

is a locale number that corresponds to a locale on the system.

indicator

is the kind of CCS number desired.

0 = is the Unisys CCS number for use with CCS_TO_CCS transliteration.

1 = is the ISO CCS number for interoperability use.

2 = is the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is the CCS number associated with the locale. If the service routine returns an error status other than OK, this value is undefined.

Error Statuses

LOCALE_NUM_TO_CCS_NUM can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 26 – ISLLOCNOTFOU
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 45 – ISLCNUMFORUN

Sample Code

This code example shows how to use LOCALE_NUM_TO_CCS_NUM to return the CCS number associated with the locale number.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii                      .
locnum    +         4                 . locale number
          islpktgen                   . generate islpkt, inc. EQUFs
.
. Messages issued by the program:
.
errmsg    'ERROR STATUS FROM BMSM$LNB2CNB' . print error msg
errmsgl   $equ      $-errmsg          . message length
zeroes    +         0                 . word of zeroes
.
$(1)                                  .
lnb2cnbex*                            .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . get pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          la        a0,locnum         . get locale number
          sa        a0,islcnvnum      . save number in packet
          sz        islindicator      . want ccs number?
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$lnb2cnb  . give control to lnb2cnb
          tnz       islerror          . was there an error?
          j         noerror           . no, continue processing
          a$print   (pf 1,errmsgl,errmsg). yes, print error msg
          j         rtn1              . jump to common code
noerror                               . process returned number
.
```

```
.   Normal processing here
.
rtn1                                    .
          er        exit$               .
          $end      lnb2cnbex           . end lnb2cnb
```

5.27. LOCALE_NUM_TO_LOCALE_NAME Service Routine

Entry Point

BMSM$LNB2LNM

Description

LOCALE_NUM_TO_LOCALE_NAME transforms a locale number to its corresponding locale name.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$LNB2LNM
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

conversion number

is a locale number that corresponds to a locale on the system.

address 1

is a pointer to the string into which the service routine should write the corresponding the locale name. This name is required for the SETLOCALE service routine. It is case-sensitive and corresponds to a LOCALE_NAME entity in the Repository for ClearPath OS 2200. If the service routine returns an error status other than OK, this value is undefined.

offset 1

is indicated by *address 1*. This is where the locale name is to begin.

0 = the name should begin on the word boundary (Q1) indicated by *address 1*.

1 = the name should begin in Q2.

2 = the name should begin in Q3.

3 = the name should begin in Q4.

character length 1

is the maximum bit length for the locale name area indicated by *address 1*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is the bit length of the locale name. If the service routine returns an error status other than OK, this value is undefined.

address 1

is a pointer to a buffer that contains the locale name. This value is set by the LOCALE_NUM_TO_LOCALE_NAME operation.

If the operation is successful, the buffer contains the name of the locale corresponding to the conversion number value.

If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 2* string is undefined.

Error Statuses

LOCALE_NUM_TO_LOCALE_NAME can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 26 -SLLOCNOTFOU
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV

Sample Code

This code example shows how to use LOCALE_NUM_TO_LOCALE_NAME to transform a locale number to its corresponding locale name.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
locnam    $res      4                 . locale name
locnum    +         6                 . locale number
          islpktgen                   . generate islpkt, inc. EQUFs
.
. Messages issued by the program:
.
errmsg    'ERROR STATUS FROM BMSM$LNB2LNM' . print error msg
errmsgl   $equ      $-errmsg          . message length
zeroes    +         0                 . word of zeroes
.
$(1)                                  .
lnb2lnmex*                            .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . get pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          lxi,u     a0,144            . locale name area bit length
          lxm,u     a0,locnam         . get addr. of locale name
          sa        a0,isllenaddr1    . save length and address
          la,u      a0,0              . get 1st char. position
          sa        a0,isloffset1     . save name of char. position
          la        a0,locnum         . load loc. number to find
          sa        a0,islcnvnum      . store loc. number to be found
          la,u      a0,islpkt         . get packet address
          lxi,u     ao,islpktlen      . get packet length
          i$bj      x11,bmsm$lnb2lnm  . give control to lnb2lnm
          tnz       islerror          . was there an error?
          j         noerror           . no, continue processing
          a$print   (pf 1,errmsgl,errmsg) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
.
. Process the returned locale name.
.
rtn1                                  .
          er        exit$             .
          $end      lnb2lnmex         . end lnb2lnm
```

5.28. NAME_AND_NUMBER Packet Interface

Entry Point

BMSM$ISLNMNB

Description

The NAME_AND_NUMBER packet interface provides a general interface for obtaining name-and-number information for various locales and CCSs. The Virtual Machine for the Java (TM) Platform on ClearPath OS 2200 can use this interface to obtain this information, which also includes the number of bits in a character for the requested locale or CCS.

Note: All locales have an associated CCS number. However, not all CCSs are associated with a locale.

Calling Sequence

```
la,u      a0,islpkt          . get address of packet
lxi,u     a0,islpktlen       . get general name-number packet length
i$bj      x11,bmsm$islnmnb   . call service routine
```

Error Statuses

NAME_AND_NUMBER can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 26 – ISLLOCNOTFOU
- 31 – ISLCCSNOTFOU
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 40 – ISLCNUMNOTUN
- 45 – ISLCNUMFORUN
- 57 – ISLINVNMNBID
- 58 – ISLINVNMLEN

Format of NAME_AND_NUMBER Packet

The following figure shows the format for the I18NLIB Service Library NAME_AND_NUMBER information packet (ISL-NMNBIP).

Note: *Not all I18NLIB service routines use all fields of this packet. As a user, you need to initialize to zero (0) any unused or reserved packet fields.*

reserved	reserved	reserved	identifier	char-bit-size	version
number					
ccs-name-length					
ccs-number					
ccs-iso-number					
ccs-sdf-number					
ccs-name+0 (characters 01-04)					
ccs-name+1 (characters 05-08)					
ccs-name+2 (characters 09-12)					
ccs-name+3 (characters 13-16)					
locale-name-length					
locale-number					
locale-name+0 (characters 01-04)					
locale-name+1 (characters 05-08)					
locale-name+2 (characters 09-12)					
locale-name+3 (characters 13-16)					

Figure 5–1. NAME_AND_NUMBER Packet Format

Fields

See the NAME_AND_NUMBER Packet Interface in the COBOL section for the list of fields in the *name_and_number* information packet.

Sample Code

This code example shows how to use NAME_AND_NUMBER to obtain CCS number information.

```
          $include  'maxr$'           . define registers
          $include  'isl-pkt-defs/msm' . define ISLNMNB packet integers
          $ascii                      .
$(0)                                  .
          islpktgen                   .
.
zeroes    +         0                 . zeroes
ucs2ccsno $equ      61                .
$(1)                                  .
start*                                .
          la        a0,(0,zeroes)     . get source for block transfer
          la        a1,(1,islpkt)     . get destination for block
transfer
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize entire packet
          la,u      a2,1              . set identifier as
          sa        a2,islid          .     CCS number input
          la,u      a0,ucs2ccsno      . set CCS number input
          sa        a0,islnumber      . save CCS number
          la,u      a0,islpkt         . get packet
```

```
            lxi,u     a0,islpktlen      . get packet length
            i$bj      x11,bmsm$nmnb     . go to the service routine
            tnz       islerror          . was there an error?
            j         check1            . no
            .
            .  Error processing here
            .
  check1                                . (check returned values)
            .
            .
            .
```

5.29. NL_LANGINFO Service Routine

Entry Point

BMSM$NLLANG

Description

NL_LANGINFO returns a string containing relevant information about the particular language or cultural area defined in the program's locale.

NL_LANGINFO requires a specific locale item number so it can determine what locale information to return.

Calling Sequence

```
I$BJ      X11,BMSM$NLLANG
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

islversion

is the packet version. You must set this field to zero (0).

islitem

is the number indicating which locale item to retrieve. (Variables equated to possible values for this field are automatically defined when `ISL-PKT-DEFS/MSM` is included.)

islchrlen1

is set to the maximum number of characters that can be written in the output buffer.

isladdr1

is the DBANK relative address of the output buffer where the service routine places the resulting string. In a locale where the locale data is not defined, the service routine returns a string to the corresponding string in the POSIX locale.

isloffset1

is the position of the first character in the output buffer (where 0=Q1, 1=Q2, 2=Q3, and 3=Q4).

Return Values

islerrorwd

contains the error structure returned by the service routine.

islresult

contains the number of characters in the resulting string if islerrorwd indicates normal completion.

In a locale where the particular locale is not defined, NL_LANGINFO returns a string to the correponding string in the POSIX locale.

The character length of the resulting string is also returned in islresult.

In all locales, NL_LANGINFO returns an error if islitem contains an invalid status value.

Sample Code

NL_LANGINFO returns a string containing information about a particular language or cultural area defined in the program's locale. This code example illustrates a call to return the full name of the fourth day in the week, based on the POSIX locale, which specifies the minimal environment for C language translation.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
          $ascii                      .
pf        $form     12,6,18           .
$(0)      .
outbufl   $equ      5                 . # of output buffer words
outbuf    $res      outbufl           . space for resulting string
          islpktgen                   . generate islpkt and equfs
.
errmsg    'ERROR RETURNED BY NL_LANGINFO' . print error message
errmdgl   $equ      $-errmsg          . message length
$(1)                                  .
atart                                 .
          la        a0,(0,(0))        . used to zero packet
          la        a1,(1,islpkt)     . get increment & pkt
```

```
                                . addr
lr,u      r1,islpktlen          . get packet length
bt        a1,0,*a0              . initialize packet
la,u      a0,isl_day_4          . item #11
sa        a0,islitem            . save into packet
la,u      a0,outbuf             . get buffer address
lxi,u     a0,outbufl*4          . get buffer char length
sa        a0,isllenaddr2        . save into packet
la,u      a0,islpkt             . get address of islpkt
lxi,u     a0,islpktlen          . get packet length
i$bj      x11,bmsm$nllang       . trans control to service
                                . routine
la        a0,(pf 1,outbufl,outbuf) . print parameter
tz        islerror              . was there an error?
la        a0,(pf 1,errmsgl,errmsg) . yes-print
                                . parameter
er        aprint$               . print appropriate
                                . message
    .
    .
    .
```

Assuming that the locale for LC_TIME is "POSIX", the following output is displayed:

```
Wednesday
```

5.30. OPEN_CCS_TO_CCS Service Routine

Entry Points

BMSM$OPENCCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$OPENCCS
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

from ccs

is an integer that contains the Unisys CCS number of the *from-string*.

to ccs

is an integer that contains the Unisys CCS number of the *to-string*.

Return Values

error

is the error structure returned by the service routine.

result

is the conversion descriptor (internal identifier for the transliteration tables). This value is passed to CCS_TO_CCS.

Sample Code

See 5.8 for sample code.

5.31. PUTENV Service Routine

Entry Point

BMSM$PUTENV

Description

PUTENV sets the current value of the environment variable for the run level.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$PUTENV
```

Setting Environment Variable Values

PUTENV places a *name=value* string into the run control information.

name

is the name of an environment variable; it is indicated by *address 1*.

value

is the value to be set for the environment variable; it is indicated by *address 2*.

In the TIP environment, this value is limited to 16 characters. If it is longer, PUTENV truncates it to 16 characters and returns the following warning:

```
EV VALUE TOO LONG
```

Trailing Spaces in Environment Variable Names and Values

Trailing spaces are significant in the names and values of environment variables. For example, a name such as 'LC_COLLATE..' is not the same as 'LC_COLLATE'. PUTENV would match 'LC_COLLATE..' with the I18N environment variable LC_COLLATE.

They can be a part of the environment variable value. Any trailing spaces in the *address 2* value are placed in the *name=value* string.

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to a string containing the environment variable name.

character length 1

is the length, in characters, of the string pointed to by *address 1*.

address 2

is a pointer to a string containing the value to be set for the environment variable.

character length 2

is the length, in characters, of the string pointed to by *address 2*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

Sample Code

This code example shows how to use PUTENV to set environment variables (EV) at the run level.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
cpw       $equ      4                 . 4 ASCII char. per word
.
. Environment variable name and value character size
. definitions:
.
lc_col_sz  $equ     $sl('LC_COLLATE') . char. length of EV
lc_ctyp_sz  $equ    $sl('LC_CTYPE')   . char. length of EV
lc_all_sz  $equ     $sl('LC_ALL')     . char. length of EV
lang_sz   $equ      $sl('LANG')       . char. length of EV
lc_col_v_sz  $equ   $sl('en_US.ASCII') . char. len. of EV value
lc_ctyp_v_sz  $equ  $sl('en_US.ASCII') . char. len. of EV value
lc_all_v_sz  $equ   $sl('en_US.ASCII') . char. len. of EV value
lang_v_sz  $equ     $sl('en_US.ASCII') . char. len. of EV value
.
. Environment variable name definitions:
.
lc_collate  'LC_COLLATE'              . EV name
lc_ctype  'LC_CTYPE'                  . EV name
lc_all    'LC_ALL'                    . EV name
lang      'LANG'                      . EV name
lc_collate_v  'en_US.ASCII'           . EV value
lc_ctype_v  'en_US.ASCII'             . EV value
lc_all_v  'en_US.ASCII'               . EV value
lang_v    'en_US.ASCII'               . EV value
msg2      'ERROR STATUS FROM BMSM$PUTENV' . print error msg
msg2l     $equ      $-msg2            . message length
msg3      'PUTENV WORKED'             . message for no error
msg3l     $equ      $-msg3            . message length
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt, inc.
                                      . EQUFs
.
$(1)                                  .
putenvex*                             .
          la        a0,(0,zeroes)     . get source for BT
          la        a1,(1,islpkt)     . get destination of BT
          lr,u      r1,islpktlen      . get length of packet
          bt        a1,,*a0           . initialize entire packet
          la,u      a0,lc_collate     . get addr. of EV name
          lxi,u     a0,lc_col_sz      . get char. size of EV
```

```
          sa        a0,isllenaddr1    . save into packet
          la,u      a0,lc_collate_v   . get addr. of EV value
          lxi,u     a0,lc_col_v_sz    . get char. size of EV value
          sa        a0,isllenaddr2    . save into packet
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$putenv   . give control to putenv
          tnz       islerror          . was there an error?
          j         noerror           . no, continue processing
          a$print   (pf 1,msg2l,msg2) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg3l,msg3) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      putenvex          . end putenv
```

5.32. SETLOCALE Service Routine

Entry Point

BMSM$SETLOC

Description

SETLOCALE is used to set category values and query locale information for a user program.

The setting of category values is indeterminate when a user program begins execution. A program calls SETLOCALE to set all or a portion of these values, as specified by the *category* and *address 1* fields.

If *address 1* is set to "_QUERY", SETLOCALE writes the name of the locale associated with the category into the data area indicated by *address 2*. The category's locale is not changed.

In a subsequent call to SETLOCALE, you can include the *address 2* string value as the *address 1* field to restore a category value.

A query of the LC_ALL category returns the locale name for the LC_COLLATE category.

Once called by the program, SETLOCALE establishes the locale for the current activity.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$SETLOC
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

category

is an integer that specifies the portion of the program's locale to be set or queried.

There are seven allowable categories: 0=LC_ALL, 1=LC_COLLATE, 2=LC_CTYPE, 3=LC_MESSAGES, 4=LC_MONETARY, 5=LC_NUMERIC, and 6=LC_TIME.

address 1

is a pointer to a string that specifies the name of a locale.

character length 1

is the length, in characters, of the requested locale name. The maximum length for a locale name is 16 characters. When dealing with locale names, SETLOCALE is case-sensitive and ignores trailing spaces.

offset 1

is the byte offset for the requested locale name. This is the location where the locale name begins.

0 = the string begins on the word boundary (Q1) indicated by *address 1*.

1 = the string begins in Q2.

2 = the string begins in Q3.

3 = the string begins in Q4.

address 2

is a pointer to a buffer for holding the resulting locale name. The pointer for *address 2* may be the same as the one for *address 1*. If this operation is not successful, the content of the string pointed to by *address 2* is undefined.

character length 2

indicates the bit length of the value indicated by *address 2*. If the SETLOCALE operation is not successful, the length of the locale name is undefined.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

address 2

If the SETLOCALE operation is successful, the area indicated by *address 2* contains the locale name that was set or queried for in *address 1*.

If the SETLOCALE operation is not successful, an error status is returned in the *error* field that indicates why it failed, the value of the *address 2* string is undefined and the program's category value is not changed.

character length 2

indicates the bit length of the value indicated by *address 2*. If the SETLOCALE operation is not successful, the length of the locale name is undefined.

Error Statuses

SETLOCALE can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – SLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 28 – ISLINVCAT
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use SETLOCALE to establish a locale for an activity.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
cpw       $equ      4                 . 4 ASCII characters per word
.
. Locale category equates:
.
lc_all_cat  $equ    0                 . LC_ALL
lc_collate_c  $equ  1                 . LC_COLLATE
lc_ctype_c  $equ    2                 . LC_CTYPE
max_loc_w  $equ     5                 .
max_loc_c  $equ     max_loc_w*cpw     . max. # of characters
                                      .    for resulting locale
          islpktgen                   . generate islpkt, inc. EQUFs
zeroes    +         0                 . word of zeroes
msg1      'ERROR STATUS FROM BMSM$SETLOC' . print error msg
msg1l     $equ      $-msg1            . message length
msg2      'SETLOC WORKED'             . message for no error
msg2l     $equ      $-msg2            . message length
locale1   'en_US.ASCII'               . locale string
locale1l  $equ      $-locale1         . char. length of string
query     '_QUERY'                    . query for locale
queryl    $equ      $-query           . char. len. of query string
default   '_DEFAULT'                  . default locale
defaultl  $equ      $-default         . char. len. of default
.
$(1)                                  .
setlocex*                             .
          la        a0,(0,zeroes)     . get source for BT
          la        a1,(1,islpkt)     . get destination of BT
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize entire pkt
          la,u      a0,lc_collate_c   . get category
          sa        a0,islcategory    . save category in pkt
          la,u      a0,0              . get byte offset of
                                      .   requested locale
          sa        a0,isloffset1     . save byte offset in pkt
          la,u      a0,locale1l       . get requested loc. length
          sa        a0,islchrlen1     . save requested loc.
                                      . length
          la,u      a0,locale1        . get addr. of requested
                                      . loc.
          sa        a0,isladdr1       . save requested locale
          la,u      a0,max_loc_c      . get char. len. of
```

```
                                    . resulting locale
          sa        a0,islchrlen2   . length to the maximum
          la,u      a0,res_loc      . get addr. of where to
                                    . save resulting locale
          sa        a0,isladdr2     . save addr. in pkt
          la,u      a0,islpkt       . get addr. of locale pkt
          lxi,u     a0,islpktlen    . get packet length
          i$bj      x11,bmsm$setloc . give control to setloc
          tnz       islerror        . was there an error?
          j         noerror         . no, continue processing
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1            . jump to common code
noerror                             .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                .
          er        exit$           .
          $end      setlocex        . end setloc
```

5.33. STRFMON Service Routine

Entry Point

BMSM$STRFMON

Description

STRFMON is used to convert numeric monetary values to a string representation based on a specified format and the locale information defined by the LC_MONETARY category.

Calling Sequence

```
I$BJ      X11,BMSM$STRFMON
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

islversion

is the packet version. You must set this field to zero (0).

islchrlen1

is the number of characters in the format string. This string describes the format of the resulting output buffer; it has two objects: plain characters and conversion specifications.

- Plain Characters: Plain characters are copied unchanged to the output buffer.

- Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

isladdr1

is the DBANK relative address of the format string.

isloffset1

is the character position of the first character in the format string (where 0=Q1, 1=Q2, 2=Q3, and 3=Q4).

islargs

is the DBANK relative address where the argument list begins. For more details, go to the Conversion Specification Table.

islchrlen2

is set to the maximum number of characters that can be written in the output buffer.

isladdr2

is the DBANK relative address of the output buffer where the service routine places the resulting string.

isloffset2

is the position of the first character in the output buffer (where 0=Q1, 1=Q2, 2=Q3, and 3=Q4).

Conversion Specification Table (STRFMON)

The following table shows the sequence of arguments in a conversion specification. Each conversion specification results in the fetching of zero or more sequential arguments that are converted and formatted.

- Each argument must be a double-precision floating point number.
- If excess arguments remain after the format string is exhausted, they are ignored.
- If the format string contains more conversion specifications than arguments, the results are unpredictable. With the C compiler, an error is returned.

Note: *The % character and the conversion character are required; all others are optional***.**

Table 5–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
1	% character	Required argument

Table 5–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
2	flags	Optional arguments
	=f	A numeric fill character. This character (1) must be representable in a single byte; (2) does not affect field width filling, which always uses the space character; and (3) is ignored unless left precision is specified. The default is the space character.
	^	Indicates that the currency amount should not be formatted with currency characters. If this flag is defined for the current locale, grouping characters are inserted.
	+ or (	Specifies the style for representing positive and negative currency amounts. Only one style can be specified. If +, the locale's equivalent of + and - are used. If (, negative amounts are enclosed in parentheses ( ). If neither style is specified, the default is +.
	!	Suppresses the currency symbol from output conversion.
	=	Specifies the alignment. If it is present, all fields are left justified.
3	field width (w)	Optional argument. A decimal digit that specifies a minimum field width in bytes. If the - flag is specified, the result is left-justified. If the - flag is not specified, the result is right-justified. The default is zero (0).
4	left precision (#n)	Optional argument. A pound sign followed by a decimal digit string that specifies the maximum number of digits expected to be formatted to the left of the radix character. It can be used to (1) keep formatted output from multiple calls to STRFMON aligned in the same columns and (2) fill unused positions with a special character. • If more than n digit positions are required, this value is ignored, and the excess positions are filled with numeric fill characters (=f). • If the grouping has not been suppressed with the ^ flag and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit. • To ensure alignment, any characters appearing before or after the number in the formatted output (such as currency or sign symbols) are padded as necessary with space characters to make their positive and negative formats an equal length.

Table 5–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
5	right precision (.p)	Optional argument. A period followed by a decimal digit that specifies the number of digits after the radix character. If the value is zero, no radix character appears . The amount being formatted is rounded to the specified number of digits before formatting occurs. If right precision is not included, a default specified by the current locale is used.
6	conversion character	Required argument
	i	This argument should use the locale's international currency format. USA example: `USD 1,234.56`
	n	This argument should use the locale's national currency format. USA example: `$1,234.56`
	%	This means no argument is converted. The entire conversion specification must be %%.

Return Values

islerrorwd

contains the error structure returned by the service routine.

islresult

contains the character length of the resulting string if the operation is successful.

Error Statuses

STRFMON can return the following error statuses in the *islerrorwd* field:

- 0-ISLOK
- 1-ISLINVPKTLEN
- 2-ISLINVPKTVER
- 3-ISLBUFSHORT
- 20 – ISLNAMINVCHR
- 21-ISLVALINVCHR
- 26-ISLLOCNOTFOU
- 30-ISLCORRUPTAB
- 34-ISLINVBMADDR
- 35-ISLINVEMENV
- 39-ISLGETENVERR

- 41-ISLASTRREAD
- 42-ISLASTRWRITE
- 43-ISLCREATBANK
- 44-ISLCHPTRESRT
- 51-ISLINVCNVSPC
- 52-ISLINVARGMTS

Sample Code

This code example shows how to use STRFMON to format a monetary value.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
          $ascii                      .
pf        $form     12,6,18           .
$(0)                                  .
fmt       $equ      'Cost is %=*#5n (%#5.0n rounded).' .
fmtlen    $equ      $sl(fmt)          . char len of string
fmtimage  $gen      fmt               . string to be formatted
args      +         (12345*-2)D       . arguments: 123.45
          +         (12345*-2)D       .            123.45
outbufl   $equ      10                . # of output buffer words
outbuf    $res      outbufl           . space for resulting string
          islpktgen                   . generate islpkt & equfs
.
errmsg    'ERROR RETURNED BY STRFMON' . print error message
errmsgl   $equ      $-errmsg          . message length
$(1)                                  .
start                                 .
          la        a0,(0,(0))        . used to zero packet
          la        a1,(1,islpkt)     . get increment & pkt addr
          lr,u      r1,islpktlen      . get packet length
          bt        a1,0,*a0          . initialize packet
          la,u      a0,fmtimage       . get addr of format string
          lxi,u     a0,fmtlen         . get char len of string
          sa        a0,isllenaddr1    . save into packet
          la,u      a0,args           . arguments address
          sa        a0,islargs        . save into packet
          la,u      a0,outbuf         . get buffer address
          lxi,u     a0,outbufl*4      . get char len of buffer
          sa        a0,isllenaddr2    . save into packet
          la,u      a0,islpkt         . get address of islpkt
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$strfmon  . transfer control to strfmon
          la        a0,(pf 1,outbufl,outbuf) . print parameter
          tz        islerror          . was there an error?
          la        a0,(pf 1,errmsgl,errmsg) . yes-print error msg
          er        aprint$           . print appropriate msg
              .
```

```
.
.
```

The output for this locale is displayed in the following format:

```
Cost is $***123.45 ($  123 rounded).
```

5.34. STRFTIME Service Routine

Entry Point

BMSM$STRFTIM

Description

STRFTIME is used to convert date and time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

Calling Sequence

```
I$BJ      X11,BMSM$STRFTIM
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

islversion

is the packet version. You must set this field to zero (0).

islchrlen1

is the number of characters in the format string. This string describes the format of the resulting output buffer; it has two objects: plain characters and conversion specifications.

- Plain Characters: Plain characters are copied unchanged to the output buffer.
- Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

isladdr1

is the DBANK relative address of the format string. For details, go to Conversion Specifications in a Format String.

islchrlen2

is set to the maximum number of characters that can be written in the output buffer.

isladdr2

is the DBANK relative address of the output buffer where the service routine places the resulting string.

isloffset1

is the position of the first character of the format string (where 0=Q1, 1=Q2, 2=Q3, and 3=Q4).

isloffset2

is the position of the first character in the output buffer (where 0=Q1, 1=Q2, 2=Q3, and 3=Q4).

isltmaddr

is the DBANK relative address of the date/time structure packet. To see this packet, go to the Date-and-Time Structures Table.

Conversion Specification Table (STRFTIME)

Each conversion specification (CS) is replaced by the appropriate characters listed in this table. These characters are determined by the current locale and by the values contained in the date-and-time structure. If a CS does not correspond to any of the ones listed in this table, the behavior is undefined.

Table 5–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%a	Weekday name, abbreviated	abday	tm_wday
%A	Weekday name, full	day	tm_wday
%b	Month name, abbreviated	abmon	tm_mon
%B	Month name, full	mon	tm_mon
%c	Appropriate date/time representation	d_t_fmt	Varies by locale
%C	Century number. To get this number, the year is divided by 100 and truncated to an integer as a decimal [00-99].	n/a	tm_year
%d	Day of month as decimal [01,31]	n/a	tm_mday
%D	Equivalent to %m/%d/%y	n/a	tm_mon, tm_mday, and tm_year.
%e	Day of month as decimal [1,31]. A single digit is preceded by a space.	n/a	tm_mday

Table 5–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%F	Equivalent to %Y-%m-%d (the ISO8601:2000 standard date format)	n/a	tm_year, tm_mon, and tm_mday
%g	Replaced by the last 2 digits of the week-based year as a decimal number [00-99].	n/a	tm_year, tm_yday, and tm_mday
%G	Replaced by the week-based year and as a decimal number.	n/a	tm_year, tm_yday, tm_mday
%h	Month name, abbreviated	abmon	tm_mon
%H	Hour on 24-hour clock as decimal [00,23]	n/a	tm_hour
%I	Hour on 12-hour clock as decimal [01,12]	n/a	tm_hour
%j	Day of year as decimal [001,366]	n/a	tm_yday
%m	Month as decimal [01,12]	n/a	tm_mon
%M	Minute as decimal [00,59]	n/a	tm_min
%n	New line character (012)	n/a	n/a
%p	Equivalent for ante-meridiem (AM) and post-meridiem (PM)	am_pm	tm_hour
%r	Time in AM and PM notation; equivalent of %I:%M:%S %p in the POSIX locale	t_fmt_ampm	Refer to individual CSs.
%R	Time in 24-hour notation (%H:%M)	n/a	tm_hour and tm_min.
%S	Second as decimal [00,60]	n/a	tm_sec
%t	Tab character	n/a	n/a
%T	Time (%H:%M:%S)	n/a	tm_hour, tm_min, and tm_sec.
%u	Weekday as decimal [1,7]	n/a	tm_wday
%U	Week number of the year as decimal [00,53] with Sunday as day 1	n/a	tm_yday and tm_wday
%V	Week number of the year as decimal [01,53] with Monday as day 1 If the week containing January 1 has four or more days in the new year, then it is considered week 1. If is does not, then it is the last week of the previous year, and the next week is week 1.	n/a	tm_year, tm_mday, and tm_yday

Table 5–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%w	weekday as decimal [0,6] with 0 representing Sunday	n/a	tm_wday
%W	Week number of the year as decimal [00,53] with Monday as day 1. All days in a new year preceding the first Sunday are considered to be in week 0.	n/a	tm_wday and tm_yday
%x	Appropriate date representation	d_fmt	Varies by locale
%X	Appropriate time representation	t_fmt	Varies by locale
%y	Year without century as decimal [00,99]	n/a	tm_year
%Y	Year with century as decimal [1998]	n/a	tm_year
%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm) or by no characters if no timezone is determinable.	n/a	tm_isdst
%Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.	n/a	tm_isdst
%%	%	n/a	n/a

Return Values

islerrorwd

contains the error structure returned by the service routine.

islresult

contains the character length of the resulting string if the operation is successful.

Error Statuses

STRFTIME can return the following error statuses in the *islerrorwd* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 30 – ISLCORRUPTAB

- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT
- 51 – ISLINVCNVSPC
- 52 – ISLINVARGMTS
- 56 – ISLINVDATTIM
- 59 – ISLSYSTIMERR

Sample Code

This code example shows how to use STRFTIME to format a date value to a string representation based on locale en_US.8859-1 (English as spoken in the US).

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
          $ascii                      .
pf        $form     12,6,18           .
$(0)                                  .
bufsize   $equ      8                 . formatted buffer size
                                      .   (words)
tbuf      $res      bufsize           . # words for buffer
str       'TODAY IS %x.'              . string to be formatted
strlen    $equ      $sl('TODAY IS %x.') . char len of string
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt and equfs
          isltmgen                    . generate isltmpkt and equfs
.
. Here are the messages issued by the sample program.
.
errmsg    'ERROR STATUS FROM BMSM$STRFTIM ' . print error msg
errmsgl   $equ      $-errmsg          . message length
$(1)                                  .
strftimeex*                           .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . get increment and pkt addr
          lr,u      r1,islpktlen      . get packet length
          bt        a1,0,*a0          . initialize packet
          la,u      a0,str            . get addr of format string
          lxi,u     a0,strlen         . get char len of string
          sa        a0,isllenaddr1    . save length and address
          sz        isloffset1        . string char position (Q1)
          la,u      a0,tbuf           . get buffer address
          lxi,u     a0,bufsize*4      . get buffer len in characters
          sa        a0,isllendddr2    . save buffer len and addr
```

```
            sz        isloffset2        . buffer character position
            ls,u      a0,isltmpkt       . get isltmpkt address
            sa        a0,isltmaddr      . save isltmpkt address
            la,u      a1,7              . month of August
            sa        a1,isltmmon,a0    . save month
            la,u      a1,22             . 22nd day of month
            sa        a1,isltmmday,a0   . save day of the month
            la,u      a1,95             . year 1995
            sa        a1,isltmyear,a0   . save year
            la,u      a0,islpkt         . get address of islpkt
            lxi,u     a0,islpktlen      . get packet length
            i$bj      X11,bmsm$strftim  . transfer control to strftime
            tnz       islerror          . was there an error?
            j         strok             . no-continue processing
            a$print   (pf 1,errmsgl,errmsg) . yes-print error msg
            j         rtn1              . jump to common code
strok                                   .
            a$print   (pf 1,bufsize, tbuf) . print out formatted str
rtn1                                     .
                .
                .
                .
```

The output for this locale is displayed in the following format:

```
TODAY IS 08/22/95.
```

5.35. STRING_COLLATE Service Routine

Entry Point

BMSM$STRCOLL

Description

STRING_COLLATE compares two character strings and returns their collating order. These strings can be up to 4096 characters long. The collating rules are determined by the current setting of the LC_COLLATE category.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$STRCOLL
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the first string containing characters to be compared.

offset 1

is a character position in the string indicated by *address 1*. This is the location of the first character to compare.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 2*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 1

is the number of characters to be compare, starting at *offset 1*.

address 2

is a pointer to the second string containing characters to be compared.

offset 2

is a character position in the buffer indicated by *address 2*. This is the location of the first character to compare.

0 = the comparison should begin on the word boundary (Q1) indicated by *address 2*.

1 = the comparison should begin in Q2.

2 = the comparison should begin in Q3.

3 = the comparison should begin in Q4.

character length 2

is the number of characters to be compared, starting at *offset 2*.

collate address

the address of a collate table that resides in the application data space.

If no collate table address is to be passed to STRING_COLLATE, this value must be set to zero (0).

If the value is zero, STRING_COLLATE finds the collate table based on the value of the LC_COLLATE category.

If the value is non-zero, STRING_COLLATE assumes that the application has called GET_COLLATE_TABLE and that the collate table resides in the data space indicated by *collate address*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is the result of the comparison. The possible values are

-1 = the first string collates before the second string.

0 = the strings collate equally.

1 = the first string collates after second string.

If error status is not OK, then result is undefined.

Error Statuses

STRING_COLLATE can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 7 – ISLENCNOTSUP
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 36 – ISLSTR12LONG
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE

- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use STRING_COLLATE to compare two character strings and return their collating order.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
cpw       $equ      4                 . 4 ASCII char. per word
.
. String and string length:
.
in_string1  'Unisys'                  . string to be transformed
in_str_len1  $equ   $sl('Unisys')     . char. length of string
.
in_string2  'unisys'                  . string to be transformed
in_str_len2  $equ   $sl('unisys')     . char. length of string
.
msg1      'ERROR STATUS FROM BMSM$STRCOLL' . print error msg
msg1l     $equ      $-msg1            . message length
.
msg2      'Unisys collates before unisys'
msg2l     $equ      $-msg2            . message length
.
msg3      'Unisys collates after unisys'
msg3l     $equ      $-msg3            . message length
.
msg4      'Unisys collates equally with unisys' .
msg4l     $equ      $-msg4            . message length
.
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt, get EQUFs
$(1)                                  .
stringcoll*                           .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . load pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . clear packet
          la,u      a0,in_string1     . get addr. of 1st string
          lxi,u     a0,in_str_len1    . get char. len. of 1st str
          sa        a0,isllenaddr1    . save addr. of 1st str len
          la,u      a0,2              . set char. posit. of 1st str
          sa        a0,isloffset1     . save offset of 1st str
          la,u      a0,in_string2     . get addr. of 2nd string
          lxi,u     a0,in_str_len2    . get char. len. of 2nd str
```

```
          sa        a0,isllenaddr2    . save addr. of 2nd str len
          la,u      a0,0              . get char. posit. of 2nd str
          sa        a0,isloffset2     . save offset of 2nd str
          la,u      a0,0              . address of collate table
          sa        a0,islcoladdr     . save collate address
.
. Two strings are compared.  The service routine determines
. the collation order based on the rules set by LC_COLLATE.
. Different locales will cause the strings to collate differently.
.
. In this example, "Unisys" will collate before "unisys" if
. the ASCII code set is used to set LC_COLLATE to the string
. "en_US.ASCII", which corresponds to the English language
. as spoken in the US.
.
. If LC_COLLATE indicates some other locale,
. then the two strings might collate in a different order.
.
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$strcoll  . give control to strcoll
          tnz       islerror          . was there an error?
          j         noerror1          . no, continue processing
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn               . jump to common code
noerror1                              .
          la        a1,islresult      . get the result
          jz        a1,collequal      . collate equally?
          te,u      a1,1              . no, 2nd str collated 1st?
          j         firstless         . no, jump
          a$print   (pf 1,msg3l,msg3) . yes, print unisys
                                      .    collates first
          j         rtn1              . jump to common code
collequal                             .
          a$print   (pf 1,msg4l,msg4) . print collate equally
          j         rtn1              . jump to common code
firstless                             .
          a$print   (pf 1,msg2l,msg2) . print Unisys collates first
rtn1                                  .
          er        exit$             .
          $end      stringcoll        . end string_collate
```

5.36. STRING_TRANSFORM Service Routine

Entry Point

BMSM$STRXFRM

Description

STRING_TRANSFORM transforms a character string into an ordering key. This string can be up to 4096 characters long. The transformation rules are determined by the current setting of the LC_COLLATE category.

Returning an Ordering Key

STRING_TRANSFORM returns a binary-sortable ordering key for the input string. The transformation uses the collation rules in the current locale.

The output buffer is filled to a length of *character length 2* with binary zeroes (0). The transformed length returned in *result* does not include the zero fill.

Updating an Ordering Key

If there are changes in the collating order of locales, the collating order string produced by STRING_TRANSFORM will be different from those produced in previous levels of the I18NLIB software. Therefore, if the ordering key string was used as an index or key in a file, it must be recreated with the updated ordering keys.

If the ordering key is not updated, programs will not find existing records, or they may create duplicate but distinct records, or they will find an incorrect record.

As a result of collation differences, previously sorted data may be out of order and errors could be received when attempting to merge or process this data. This would occur whether or not the collation order key was saved.

Accepting and Transforming Legal Characters

Characters are transformed into their corresponding ordering key values according to the collation rules specified in the LC_COLLATE section of the locale definition file.

- STRING_TRANSFORM accepts all characters in an input string as legal. This includes character codes that are not specified in the locale's collation rules.
- STRING_TRANSFORM assigns these characters to the end of the collating sequence in the order of their binary codes.

Sizing the Output Buffer

The output buffer must large enough to hold the transformed string. The TRANSFORM_LENGTH service routine calculates the maximum size needed.

Comparing Output Buffers

When comparing STRING_TRANSFORM results, the following rules apply to the buffers being compared:

- Both buffers should be the same length, *or*
- Substrings should be used. The byte length should be the one returned by STRING_TRANSFORM.
- If the two results from the STRING_TRANSFORM calls are not the same, the input strings are not equal.

Distinguishing Use of STRING_TRANSFORM and STRING_COLLATE

The result of comparing the output buffers of two calls to STRING_TRANSFORM for two strings is the same as if the two strings were passed to the STRING_COLLATE service routine. Both STRING_TRANSFORM and STRING_COLLATE provide for string collating, but the difference is

- STRING_TRANSFORM should be used if the same data items will be used in many comparisons. This option is cheaper, because the transformation costs are paid only once.
- STRING_COLLATE should be used if you expect a small number of comparisons to be performed within an application, or if each data item is used only once in a set of comparisons.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$STRXFRM
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the string containing the characters to be converted.

offset 1

is a character position in the string indicated by *address 1*. This is the location of the first character to be converted.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 2*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 1

is the number of characters to be converted, starting at *offset 1*.

address 2

is a pointer to the buffer in which the service routine places the converted character string.

offset 2

is a character position in the buffer indicated by *address 2*. This is where the converted string is to begin.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 2*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 2

is the maximum number of characters available in the buffer indicated by *address 2*. This is where the converted string is to be stored.

collate address

the address of a collate table that resides in the application data space. If no collate table address is to be passed to the service routine, this value must be set to zero (0).

- If the value is zero, STRING_TRANSFORM finds the collate table based on the value of the LC_COLLATE category.
- If the value is non-zero, STRING_TRANSFORM assumes that the application has called GET_COLLATE_TABLE, and the collate table resides in the data space indicated by *collate-address*.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

address 2

is a pointer to a buffer that contains the transformed string. This value is set by the STRING_TRANSFORM operation.

- If the operation is successful, the area indicated by *address 2* contains the ordering key generated from the string indicated by *address 1*.
- If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 2* string is undefined.

result

is the bit length of the transformed string.

Error Statuses

STRING_TRANSFORM can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 7 – ISLENCNOTSUP
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 36 – ISLSTR12LONG
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code routine shows how to use STRING_TRANSFORM to convert a character string to an ordering key.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
bicldes1  $form     14,4,2,16         . form for BICL descriptor 1
bicldes2  $form     14,4,1,1,16       . form for BICL descriptor 2
.
$(0)                                  .
          $ascii                      .
bufsize   $equ      9                 . transform buffer size
tbuf1     $res      bufsize           . # words for transform
                                      . buffer
tbuf2     $res      bufsize           . # words for 2nd buffer
str1      'ABCD'                      . string to be transformed
```

```
strlen1   $equ      $sl('ABCD')       . char. length of string
str2      'abcd'                      . string to be transformed
strlen2   $equ      $sl('abcd')       . char. length of string
no_bits   $res      1                 . save # of bits from strxfrm
          islpktgen                   . generate islpkt, inc. EQUFs
.
. Messages issued by this code routine:
.
errmsg    'ERROR STATUS FROM BMSM$STRXFRM' . print error msg
errmsgl   $equ      $-errmsg          . message length
.
ltmsg     'STRING 2 IS LESS THAN STRING 1' . "less than" msg
ltmsgl    $equ      $-ltmsg           . message length
.
gtmsg     'STRING 2 IS GREATER THAN STRING 1' . "greater than" msg
gtmsgl    $equ      $-gtmsg           . message length
.
eqmsg     'THE STRINGS ARE EQUAL '    . equal message
eqmsgl    $equ      $-eqmsg           . message length
$(1)                                  .
bicldes   bicldes1  0,a1,0,0          . descriptor word 1
          bicldes2  0,a0,1,0,0        . descriptor word 2
.
strxfrmex*                            .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . load pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          la,u      a0,str1           . get addr. of 1st string
          lxi,u     a0,strlen1        . get char. length of str
          sa        a0,isllenaddr1    . save length and address
          la,u      a0,0              . get position of 1st char.
                                      .    to transform
          sa        a0,isloffset1     . save string char. position
          la,u      a0,tbuf1          . get buffer address
          lxi,u     a0,bufsize*4      . get buffer length
          sa        a0,isllenaddr2    . save buffer len.and addr.
          la,u      a0,0              . get buffer char. position
          sa        a0,isloffset2     . save buffer char. position
          la,u      a0,0              . get collate table address
          sa        a0,islcoladdr     . save collate table addr.
          la,u      a0,islpkt         . get transform pkt address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$strxfrm  . give control to strxfrm
          tnz       islerror          . was there an error?
          j         str1ok            . no, continue processing
          a$print   (pf 1,errmsgl,errmsg) . yes, print error msg
          j         rtn1              . jump to common code
str1ok                                .
          l         a1,islresult      . get # of bits in str 1
          sa        a1,no_bits        . save # of bits from str 1
          la        a0,(0,zeroes)     . prepare to initialize pkt
```

```
          la        a1,(1,islpkt)     . load pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . initialize packet
          la,u      a0,str2           . get addr. of 2nd string
          lxi,u     a0,strlen2        . get char. length of string
          sa        a0,isllenaddr1    . save length and addr.
          la,u      a0,0              . get position of 1st char.
                                      .    to transform
          sa        a0,isloffset1     . save string char. position
          la,u      a0,tbuf2          . get buffer address
          lxi,u     a0,bufsize*4      . get buffer length
          sa        a0,isllenaddr2    . save buffer length & addr.
          la,u      a0,0              . get buffer char. position
          sa        a0,isloffset2     . save buffer char. position
          la,u      a0,0              . get collate table addr.
          sa        a0,islcoladdr     . save collate table addr.
          la,u      a0,islpkt         . get addr. of transform pkt
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$strxfrm  . give control to strxfrm
          tnz       islerror          . was there an error?
          j         biclcompa         . no, compare strings
          a$print   (pf 1,errmsg1,errmsg) . yes, print error msg
          j         rtn1              . jump to common exit
biclcompa                             . ascending sequence
          lr        r1,no_bits        . get # of bits in string 1
          la        a1,islresult      . get # of bits in string 2
          tg        a1,no_bits        . use greater len.for compare
          lr        r1,islresult      . get bit length for str 2
          la,u      a1,tbuf1          . get str1 bit offset & addr.
          la,u      a0,tbuf2          . get str2 bit offset & addr.
          bicl      bicldes           . compare the strings
          jo        eq                . strings are equal (DB19=1)
          jc        gt                . if carry, str2 > str1
.
. The collation rules of the locale in force have indicated
. that string 2 is less than string 1.
.
          lt                          . string 2 < string 1
          a$print   (pf 1,ltmsgl,ltmsg) . print "less than" msg
          j         rtn1              . jump to common exit
.
. The collation rules of the locale in force have indicated
. that string 2 is greater than string 1.
.
          gt                          . string 2 > string 1
          a$print   (pf 1,gtmsgl,gtmsg) . print "greater than" msg
          j         rtn1              . jump to common exit
.
. The collation rules of the locale in force have indicated
. that string 2 equals string 1.
.
          eq                          . string 2 equals string 1
```

```
            a$print   (pf 1,eqmsgl,eqmsg) . print "equal" msg
  rtn1                                    .
            er        exit$
            $end      strxfrmex         . end strxfrm
```

5.37. STRPTIME Service Routine

Entry Point

BMSM$STRPTIM

Description

STRPTIME is used to convert date and time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

Calling Sequence

```
I$BJ      X11,BMSM$STRPTIM
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

islversion

is the packet version. You must set this field to zero (0).

islchrlen1

is the number of characters in the string to be converted.

isladdr1

is the DBANK relative address of the character string to be converted.

islchrlen2

is the number of characters in the format string. This string describes the format of the resulting output buffer. It includes zero or more directives, which consist of one or more white-space characters and either a plain character or a conversion specification.

Directives: The directives are the STRPTIME execution rules. For more details, go to the List of Directives in a Format String.

Plain Character: A plain character is copied unchanged to the output buffer.

Conversion Specification: A conversion specification consists of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

isladdr2

is the DBANK relative address of the format string.

isloffset1

is the position of the first character to be converted (where 0=Q1, 1=Q2, 2=Q3, and 3=Q4).

isloffset2

is the character position of the first format directive (where 0=Q1, 1=Q2, 2=Q3, and 3=Q4).

isltmaddr

is the DBANK relative address of the date/time structure packet. To see this packet, go to the Date-and-Time Structures Table.

See Appendix B for the Conversion Specification Table and Table 5–1 for the Date-and-Time Structures Table.

Return Values

islerrorwd

contains the error structure returned by the service routine.

Error Statuses

STRPTIME can return the following error statuses in the *islerrorwd* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34 – ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK

- 44 – ISLCHPTRESRT
- 51 – ISLINVCNVSPC
- 54 – ISLCHRNOTMCH
- 56 – ISLINVDATTIM

Sample Code

This code example shows how to use STRPTIME to convert date string characters to values based on locale en_US.8859-1 (English as spoken in the US).

```
          $include  'maxr$'           .
          $include  'eru$ '           .
          $include  'isl-pkt-defs/msm' .
          $ascii                      .
pf        $form     12,6,18           .
$(0)                                  .
instr     'TODAY IS 08/22/95.  '      . string to be converted
instrlen  $equ      $sl('Today is 08/22/95.  ') . char len of instr
formstr   'TODAY IS %x.'              . format string
formstrlen $equ     $sl('Today is %x.') . char len of format str
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt and equfs
          isltmgen                    . generate isltmpkt and equfs
.
. Here are the messages issued by the sample program.
.
errmsg    'ERROR STATUS FROM BMSM$STRPTIM' . print error message
errmsgl   $equ      $-errmsg          . message length
okmsg     'STRPTIME WORKED OK'        . OK message
okmsgl    $equ      $-okmsg           . message length
$(1)                                  .
strptimeex*                           .
          la        a0,(0,zeroes)     . prepare to intialize pkt
          la        a1,(1,islpkt)     . get increment & pkt addr
          lr,u      r1,islpktlen      . get packet length
          bt        a1,0,*a0          . initialize packet
          la,u      a0,instr          . get addr of input string
          lxi,u     a0,instrlen       . get char len of in_string
          sa        a0,isllenaddr1    . save length and address
          sz        isloffset1        . in_string char position
          la,u      a0,formstr        . get addr of format string
          lxi,u     a0,formstrlen     . get char len of format str
          sa        a0,issllnaddr2    . save length and address
          sz        isloffset2        . format string char position
          la,u      a0,isltmpkt       . get isltmpkt address
          sa        a0,isltmaddr      . save isltmpkt address
          la,u      a0,islpkt         . get address of islpkt
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$strptim  . transfer control to strptime
          tnz       islerror          . was there an error?
          j         strok             . no-continue processing
```

```
          a$print   (pf 1,errmsgl,errmsg) . yes-print error msg
          j         rtn1              .
strok                                 .
          a$print   (pf 1,okmsgl,okmsg) .
rtn1                                  .
              .
              .
              .
```

The significant fields of the date/time structure packet for this locale are filled in as follows:

```
isltmmday   22
isltmmon     7
isltmyear   95
```

5.38. TOLOWER Service Routine

Entry Point

BMSM$TOLOWER

Description

TOLOWER converts characters in a string to their corresponding lowercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$TOLOWER
```

Handling Characters with No Corresponding Lowercase Representation

If a character in the string has no corresponding lowercase representation, the character is written to the output buffer and is not altered.

See TOUPPER Handling Character Codes

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the string containing the characters to be converted.

offset 1

is a character position in the string indicated by *address 1*. This is the location of the first character to convert.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 1*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 1

is the number of characters to be converted, starting at *offset 1*.

address 2

is a pointer to the buffer in which the service routine places the converted character string.

The output buffer can be the same as the input buffer; it must be at least as large as the input string.

offset 2

is a character position in the buffer indicated by *address 2*. This is where the converted string is to begin.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 2*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 2

is the maximum number of characters available in the buffer indicated by *address 2*. This is where the converted string is to be stored.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

address 2

is a pointer to a buffer that contains the lowercased string. This value is set by the TOLOWER operation.

If the operation is successful, the area indicated by *address 2* contains the characters from the string indicated by *address 1*.

If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 2* string is undefined.

result

is the bit length of the converted string indicated by *address 2*.

Error Statuses

TOLOWER can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 17 – ISLOVERLAPOP
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34-ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code routine shows how to use TOLOWER to convert an input character string to lowercase alphabetic characters.

```
            $include  'maxr$'           .
            $include  'eru$'            .
            $include  'isl-pkt-defs/msm' .
  pf        $form     12,6,18           .
  .
```

```
$(0)                                    .
          $ascii                        .
a_string  'string value'                . define a string
strlen    $sl('string value')           . char. length of string
outbuf    $res      5                   . tolower output buffer
outlen    $equ      20                  . output length in char.
.
msg1      'ERROR STATUS FROM BMSM$TOLOWER' . print error msg
msg1l     $equ      $-msg1              . message length
.
msg2      'TOLOWER WORKED'              . message for no error
msg2l     $equ      $-msg2              . message length
.
zeroes    +         0                   . word of zeroes
          islpktgen                     . generate islpkt, inc. EQUFs
.
$(1)                                    .
tolowerex*                              .
          la        a0,(0,zeroes)       . prepare to initialize pkt
          la        a1,(1,islpkt)       . load pkt addr. & increment
          lr,u      r1,islpktlen        . get packet length
          bt        a1,,*a0             . initialize packet
          la,u      a0,a_string         . get address of string
          lxi       a0,strlen           . get length of string
          sa        a0,isllenaddr1      . save string length & addr.
          la,u      a0,0                . get byte offset
          sa        a0,isloffset1       . save byte offset of string
          la,u      a0,outbuf           . get output buffer addr.
          lxi,u     a0,outlen           . get max. possible length
          sa        a0,isllenaddr2      . save buffer address
          la,u      a0,0                . get byte offset of char.
          sa        a0,isloffset2       . save offset
          la,u      a0,islpkt           . get packet address
          lxi,u     a0,islpktlen        . get packet length
          i$bj      x11,bmsm$tolower    . give control to tolower
          tnz       islerror            . was there an error?
.
. TOLOWER was successful.  The output buffer contains the
. converted string.  The bit length is saved in the result word
. of the basic mode MASM service routine packet.
.
          j         noerror             . no, continue processing
          a$print   (pf 1,msg1l,msg1)   . yes, print error msg
          j         rtn1                . jump to common code
noerror                                 .
          a$print   (pf 1,msg2l,msg2)   . print no error msg
rtn1                                    .
          er        exit$               .
          $end      tolowerex           . end tolower
```

5.39. TOUPPER Service Routine

Entry Point

BMSM$TOUPPER

Description

TOUPPER converts characters in a string to their corresponding uppercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

Calling Sequence

```
LA,H1     A0,interface-packet-length
LA,H2     A0,address-of-interface-packet
I$BJ      X11,BMSM$TOUPPER
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

address 1

is a pointer to the string containing the characters to be converted.

offset 1

is a character position in the string indicated by *address 1*. This is the location of the first character to convert.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 1*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 1

is the number of characters to be converted, starting at *offset 1*.

address 2

is a pointer to the buffer in which the service routine places the converted character string.

The output buffer can be the same as the input buffer; it must be at least as large as the input string.

offset 2

is a character position in the buffer indicated by *address 2*. This is where the converted string is to begin.

0 = the conversion should begin on the word boundary (Q1) indicated by *address 2*.

1 = the conversion should begin in Q2.

2 = the conversion should begin in Q3.

3 = the conversion should begin in Q4.

character length 2

is the maximum number of characters available in the buffer indicated by *address 2*. This is where the converted string is to be stored.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

address 2

is a pointer to a buffer that contains the uppercased string. This value is set by the TOUPPER operation.

If the operation is successful, the area indicated by *address 2* contains the characters from the string indicated by *address 1*.

If the operation is not successful, an error status is returned that indicates why it failed, and the value of the *address 2* string is undefined.

result

is the bit length of the converted string indicated by *address 2*.

Error Statuses

TOUPPER can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 3 – ISLBUFSHORT
- 17 – ISLOVERLAPOP
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 34-ISLINVBMADDR
- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code routine shows how to use TOUPPER to convert an input character string to uppercase alphabetic characters.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
$(0)                                  .
          $ascii
a_string  'string value'              . define a string
strlen    $sl('string value')         . char. length of string
outbuf    $res      5                 . toupper output buffer
outlen    $equ      20                . output length in char.
.
msg1      'ERROR STATUS FROM BMSM$TOUPPER' . print error msg
msg1l     $equ      $-msg1            . message length
.
msg2      'TOUPPER WORKED'            . message for no error
msg2l     $equ      $-msg2            . message length
.
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt, inc. EQUFs
.
```

```
$(1)                                .
toupperex*                          .
          la        a0,(0,zeroes)   . prepare to initialize pkt
          la        a1,(1,islpkt)   . load pkt addr. & increment
          lr,u      r1,islpktlen    . get packet length
          bt        a1,,*a0         . initialize packet
          la,u      a0,a_string     . get address of string
          lxi       a0,strlen       . get length of string
          sa        a0,isllenaddr1  . save string length & addr.
          la,u      a0,0            . get byte offset
          sa        a0,isloffset1   . save byte offset of string
          la,u      a0,outbuf       . get output buffer address
          lxi,u     a0,outlen       . get max. possible length
          sa        a0,isllenaddr2  . save buffer address
          la,u      a0,0            . get byte offset of char.
          sa        a0,isloffset2   . save offset
          la,u      a0,islpkt       . get packet address
          lxi,u     a0,islpktlen    . get packet length
          i$bj      x11,bmsm$toupper  . give control to toupper
          tnz       islerror        . was there an error?
.
. TOUPPER was successful.  The output buffer contains the
. converted string.  The bit length of the converted string is
. saved in the result word of the BMSM service routine packet.
.
          j         noerror         . no, continue processing
          a$print   (pf 1,msg1l,msg1) . yes, print error msg
          j         rtn1            . jump to common code
noerror                             .
          a$print   (pf 1,msg2l,msg2) . print no error msg
rtn1                                .
          er        exit$           .
          $end      toupperex       . end toupper
```

5.40. TRANSFORM_LENGTH Service Routine

Entry Point

BMSM$TRANLEN

Description

TRANSFORM_LENGTH performs these tasks:

- Calculates the maximum length of the ordering key that STRING_TRANSFORM can create for a given string length.
- Determines the maximum size that will be required for the output buffer when using STRING_TRANSFORM. The size that is returned will depend on the current setting of the LC_COLLATE category. It may be several times larger than the value that was passed.

Calling Sequence

```
LA,H1      A0,interface-packet-length
LA,H2      A0,address-of-interface-packet
I$BJ       X11,BMSM$TRANLEN
```

Interface Packet Field Settings

Here are the required or allowable settings for the specified interface packet fields (see 5.1 for a description of the basic mode MASM interface packet). All other interface packet fields must be set to binary zero (0).

version

is the packet version. You must set this field to zero (0).

character length 1

is the length of a string to be transformed.

collate table address

is either zero (0) or the Dbank relative address of the location of the collating table acquired by GET_COLLATE_TABLE. If the value is zero (0), the locale used is the one identified by the LC_COLLATE category.

Return Values

error

is the error structure returned by the service routine. See 5.2 for a description of the *error* field and 5.3 for an explanation of the error statuses.

result

is the maximum bit length that will be required by STRING_TRANSFORM to hold a transformed string of the length specified by character length 1. If error is not OK, result is undefined.

Error Statuses

TRANSFORM_LENGTH can return the following error statuses in the *error* field:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 34 – ISLINVBMADDR

- 35 – ISLINVEMENV
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code routine shows how to use TRANSFORM_LENGTH to determine the length of the output buffer returned by STRING_TRANSFORM for a specific length string.

```
          $include  'maxr$'           .
          $include  'eru$'            .
          $include  'isl-pkt-defs/msm' .
pf        $form     12,6,18           .
.
$(0)                                  .
          $ascii                      .
a_string  'stringvalue'               . define a string
.
msg2      'ERROR STATUS FROM BMSM$TRANLEN' . print error msg
msg2l     $equ      $-msg2            . message length
.
msg3      'TRANSFORM_LENGTH WORKED'   . message for no error
msg3l     $equ      $-msg3            . message length
.
zeroes    +         0                 . word of zeroes
          islpktgen                   . generate islpkt, inc. EQUFs
$(1)                                  .
tranlenex*                            .
          la        a0,(0,zeroes)     . prepare to initialize pkt
          la        a1,(1,islpkt)     . get pkt addr. & increment
          lr,u      r1,islpktlen      . get packet length
          bt        a1,,*a0           . clear entire packet
          la,u      a0,6              . load character length
          sa        a0,islchrlen1     . save length in packet
          la,u      a0,islpkt         . get packet address
          lxi,u     a0,islpktlen      . get packet length
          i$bj      x11,bmsm$tranlen  . give control to tranlen
          tnz       islerror          . was there an error?
          j         noerror           . no, A1 contains len. in bits
          a$print   (pf 1,msg2l,msg2) . yes, print error msg
          j         rtn1              . jump to common code
noerror                               .
          a$print   (pf 1,msg3l,msg3) . print no error msg
rtn1                                  .
          er        exit$             .
          $end      tranlenex         . end transform_length
```

Section 6
Extended Mode MASM

This section describes the I18NLIB service routines available for the Extended Mode MASM language. It also provides the information about the INCLUDE elements that define the structures and EQUs values used by the service routines.

6.1. Extended Mode MASM Error Structure

Format of Error Status Word

Values

Severity

is a 3-bit integer value with the severity of the error.

0 Successful; the operation succeeded.

2 Informational; the operation did not succeed. The *error-num* field contains more detailed information.

3 Warning; the operation did not succeed. The *error-num* field contains more detailed information.

7 Major error; the operation did not succeed. The *error-num* field contains more detailed information.

Component-id

is a 15-bit unsigned integer with the I18NLIB component-id used by ELMS. This is non-zero even when the service routine executes successfully.

Error-num

is an 18-bit unsigned integer with the error number. EQUs for Error-num values are located in the file SYSLIB*PROC$.ISL-ERR-DEFS/MSM. The value 0 indicates that the service routine succeeded.

Error Statuses: ISL-ERR-DEFS/MSM

The following equates can be used to interpret the error statuses returned by the basic and extended mode MASM service routines. They can be found in file SYSLIB*PROC$., element ISL-ERR-DEFS/MSM.

$EQU	Service Routine	$EQU	Service Routine
0	ISLOK OK (no error)	32	ISLFROMEQTO From CCS Equals To CCS
1	ISLINVPKTLEN Invalid Packet Length	33	ISLBANKFULL CCS Bank Full
2	ISLINVPKTVER Invalid Packet Version	34	ISLINVBMADDR Invalid Basic Mode Address
3	ISLBUFSHORT Buffer Too Short	35	ISLINVEMENV Invalid Extended Mode Environment
4	ISLFRCCSNDEF From CCS Not Defined	36	ISLSTR12LONG String 1 Too Long
5	ISLTOCCSNDEF To CCS Not Defined	37	ISLSTR22LONG String 2 Too Long
7	ISLENCNOTSUP Encoding Not Supported	38	ISLPUTENVERR PUT$ENV Error
8	ISLENCBUFFAI Encoding of To Buffer Failed	39	ISLGETENVERR GET$ENV Error
13	ISLBADCHAR Bad Character	40	ISLCNUMNOTUN CCS Number Not Unique
14	ISLBADDESC Bad Descriptor	41	ISLASREADERR ASTORE Read Error
15	ISLTOOMANPAR (EMSM only) Too Many Parameters	42	ISLASWRITERR ASTORE Write Error
16	ISLTOOFEWPAR (EMSM only) Too Few Parameters	43	ISLCRTBNKERR Create Bank Error
17	ISLOVERLAPOP Overlapping Operands	44	ISLCPTRSTERR Checkpoint Restart Error
20	ISLNAMINVCHA EV Name Invalid Char	45	ISLCNUMFORUN CCS Number Format Undefined
21	ISLVALINVCHA EV Value Invalid Char	50	ISLINVLOCITM Invalid Locale Item

$EQU	Service Routine	$EQU	Service Routine
22	ISLAREAFULL EV Area Full	51	ISLINVCNVSPC Invalid Conversion Spec
23	ISLNAMILLTIP EV Name Illegal for TIP	52	ISLINVARGMTS Invalid Argument Value
24	ISLNAMNOTFOU EV Name Not Found	53	ISLTOOFEWARG Too Few Arguments
25	ISLVALTOOLON EV Value Too Long	54	ISLCHRNOTMCH Descriptor and Input Chr Don't Match
26	ISLLOCNOTFOU Locale Not Found	55	ISLUNSPMODCS Modified Conversion Spec Not Supported
27	ISLNOTINTER Not a Unisys Intermediate CCS	56	ISLINVDATTIM Invalid Date/Time Value
28	ISLINVCAT Invalid Category	57	ISLINVNMNBID Invalid NMNB Packet Identifier
29	ISLCATNOTSET Category Not Set	58	ISLINVNMLEN Invalid Name Length in NMNB
30	ISLCORRUPTAB Corrupted Table	59	ISLSYSTIMERR SYSTEM$TIME Error
31	ISLCCSNOTFOU CCS Not Found		

6.2. Extended Mode MASM Parameter Format

All extended mode MASM service routines use the standard calling sequence.

Three Words for Each Parameter

Word 0: Control and Descriptive Information
Word 1: Virtual Address of Data Area
Word 2: Length of the Data Area

Layout of Word 0 for an Ordinary Parameter

```
Bits     0 1 2 3 4 5      17 18     26 27 28 29 30        35
   Word 0:  I P L U S O ----- O O ----- O  E  R  W  B B B B B B
```

Field Names and Descriptions for Word 0

I = Immediate

The parameter value is located in words 1 and 2.

P = Pointer

The parameter in word 1 is a pointer (virtual address) to a secondary parameter list. None of the I18NLIB service routines use a secondary parameter list.

The I and P fields together determine the type of parameter.

(I,P) = parameter type
(0,0) = ordinary data
(0,1) = parameter list
(1,0) = immediate data

L = Length Verification Control Bit

If L = 1: The length field, word 2, is in source verification of read/write access.

If L = not 1: Target access is granted to the entire bank that covers the virtual address (VA) in word 1.

U = Indicates meaning of the length field

If U = 0, the length field is in words.

If U = 1, the length field is in bits.

S = End bit for secondary parameter list

S = zero (0), except in the final parameter descriptor of a secondary parameter list.

0 = Reserved

O must be set to zero (0).

E, R, W = Execute, Read, and Write

E, R, and W indicate the access permissions granted to the called module for this parameter.

B = Start bit for data item

B indicates where the data item starts. This item is within the virtual address (VA) in word 1.

6.3. Extended Mode MASM CBEP Elements

I18NLIB provides a common-bank entry-point (CBEP) $INCLUDE element that defines all of the Extended Mode MASM service routines entry points. This element is named CBEP$$ISL-EM and is located in the file SYSLIB*PROC$. If this element is not included in the assembly of the MASM routine that calls an I18NLIB service routine, I18NLIB also provides a collector relocatable CBEP element and a LINK object module CBEP element that is used to resolve the I18NLIB service routine reference.

6.4. Date/Time Structures (STRFTIME and STRPTIME)

The date/time structure for these two service routines is a nine-word integer packet.

Table 6–1. STRFTIME and STRPTIME Date/Time Structures

This word	Contains or indicates	And must be within this range
0	Number of seconds after the minute. The upper limit is 60 for handling "leap seconds."	[0, 60]
1	Number of minutes after the hour.	[0, 59]
2	Number of hours since midnight.	[0, 23]
3	Day of the month.	[1, 31]
4	Number of months since January. If the current month is January, this number is 0.	[0, 11]
5	Number of years since 1900 A -1 in this field indicates the year 1899. A 100 in this word indicates the year 2000.	[-1899, 8099]
6	Number of days since Sunday. If the current day is Sunday, this number is 0.	[0, 6]
7	Number of days since January 1. If the current day is January 1, this number is 0.	[0,365]
8	Status of Daylight Savings Time positive, DST in effect; 0, DST not in effect, and negative, no DST information available	language-specific

6.5. CCS_NAME_TO_CCS_NUM Service Routine

Entry Point

EMSM$CNM2CNB

Description

CCS_NAME_TO_CCS_NUM transforms a coded character set (CCS) name to its corresponding CCS number.

Calling Sequence

```
cnm2cnbva +       EMSM$CNM2CNB      . VA of CCS_NAM_TO_CCS_NUM
             LA,U    A0,2              . two parameters in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,cnm2cnbva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the CCS_NAME_TO_CCS_NUM parameters and the required input:

ccs-name

is the CCS name. This name is case-sensitive and must correspond to (1) a CCS_NAME entity in the Repository for ClearPath OS 2200 and (2) a CCS that is active on the system.

Parameter Type: Ordinary Data
Required Access: Read

ccs-number-format

is the kind of CCS number desired. The possible values are

0 = the Unisys CCS number used with the CCS-TO-CCS transliteration.

1 = the ISO CCS number for interoperability use.

2 = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records.

Parameter Type: Immediate Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the CCS number corresponding to the specified CCS name. If the service routine returns an error status other than OK, the value for Register A1 is undefined.

Error Statuses

CCS_NAME_TO_CCS_NUM can return the following error statuses in Register A0:

Error Code	Status Name	Description
0	ISLOKx	OK (No Error)
15	ISLTOOMANPAR	Too Many Parameters
16	ISLTOOFEWPAR	Too Few Parameters
31	ISLCCSNOTFOU	CCS Not Found

Error Code	Status Name	Description
45	ISLCNUMFORUN	CCS Number Format Undefined Error

Sample Code

This code example shows how to use CCS_NAME_TO_CCS_NUM to transform a CCS name to its corresponding CCS number. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
$(1)                                  .
          $lit                        . literals to loc. counter 1
cnm2cnbva +         EMSM$CNM2CNB      . VA of cnm2cnb
ccsnam    'ISO8859-1'                 . ccs name
ccsnamlen $equ      $SL('ISO8859-1')  . char. length of ccs name
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
.
. Skeleton of parameter list passed to EMSM$CNM2CNB:
.   Parameter 1: ccsnam, ordinary data (OD), read access
.   Parameter 2: numform, immediate data (ID)
.
parlskel                              .
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         9*ccsnamlen       . bit length of string
          pardes    2,0,0,0,0,0,0,0   . parameter 2 descriptor
          +         2                 . SDF number format
          +         0                 . immediate value of 0
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
callcnm2cnb*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . get length of parlist
          la,u      a1,parlskel       . get sender bank offset
                                      .   (parameter 1)
          la        a2,x10            . get rcvr bnk offset (ALS)
          lxsi,u    a1,1              . set sndr increment (short)
          lxsi,u    a2,1              . set rcvr increment (short)
          bt        a2,b1,*a1,b0,*0   . move parlist to ALS
          sbu       b0,a1             . save code bank L,BDI in A1
          aa,u      a1,ccsnam         . create VA of ccs name
          sa        a1,*1,x10,b1      . save VA in parlist
          sbu       b1,a1             . save L,BDI of ALS in A1
          aa,u      a1,x10            . create parlist VA for SCS
.
. Clear Register A2 and A3 for the standard calling sequence.
```

```
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,2              . number of parameters
          lx,u      x2,cnm2cnbva      . get entry point
          call      0,x2,b0           . xfer control to cnm2cnb
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. CCS_NAME_TO_CCS_NUM was successful.
.
processerr                          . error processing
.
. Error processing code goes here.
.
          j         comrtn            . jump to common return
comrtn                              .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
          $end                        . end ccsnam2ccsnum
```

6.6. CCS_NUM_TO_CCS_NAME Service Routine

Entry Point

EMSM$CNB2CNM

Description

CCS_NUM_TO_CCS_NAME transforms a coded character set (CCS) number to its corresponding CCS name.

Calling Sequence

```
cnb2cnmva +       EMSM$CNB2CNM      . VA of CCS_NUM_TO_CCS_NAM
            LA,U    A0,3              . three parameters in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,cnb2cnmva      . load address of service routine
            CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the CCS_NUM_TO_CCS_NAME parameters and the required input:

ccs-number

is the CCS number. The CCS must participate in a CCS-TO-CCS conversion that is active on the system.

Parameter Type: Ordinary Data
Required Access: Read

ccs-number-format

is the kind of CCS number desired. The possible values are

0 = the Unisys CCS number used with the CCS-TO-CCS transliteration.

1 = the ISO CCS number for interoperability use.

2 = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records.

Parameter Type: Immediate Data
Required Access: Read

ccs-name

is the buffer into which the service routine should write the corresponding CCS name. This name is case-sensitive and corresponds to a CCS_NAME entity in the Repository for ClearPath OS 2200. If the service routine returns an error status other than OK, this value is undefined.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the bit length of the CCS name. If the service routine returns an error status other than OK, the value for Register A1 is undefined.

Error Statuses

CCS_NUM_TO_CCS_NAME can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 31 – ISLCCSNOTFOU
- 40 – ISLCNUMNOTUN
- 45 – ISLCNUMFORUN

Sample Code

This code example shows how to use CCS_NUM_TO_CCS_NAME to transform a CCS number to its corresponding CCS name. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
$(1)                                   .
           $lit                        . literals to loc. counter 1
cnb2cnmva +         EMSM$CNB2CNM       . VA of cnb2cnm
ccsnam    $res      4                  . ccs name
ccsnamlen $equ      4                  . max. length of ccs name
ccsnum    +         35                 . ccs number
pardes    $form     2,1,1,23,1,1,1,6   . IP,L,U,S & Res.(0),E,R,W,B
.
. Skeleton of parameter list passed to EMSM$CNB2CNM:
.   Parameter 1: ccsnum, ordinary data (OD), read access
.   Parameter 2: ccsnam, OD, read write access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of number
           +         36                . bit length of number
           pardes    2,0,0,0,0,0,0,0   . parameter 2 descriptor
           +         2                 . SDF number format
           +         0                 . immediate value of 0
           pardes    0,0,1,0,0,1,1,0   . parameter 2 descriptor
           +         0                 . VA of string
           +         9*ccsnamlen       . bit length of string
parlstlen $equ      $-parlskel         . length of parameter list
als_size  $equ      parlstlen          . ALS frame size
.
callcnb2cnm*                           .
           buy       *als_size,*x10,b1 . buy ALS frame
           lr,u      r1,parlstlen      . get length of parlist
           la,u      a1,parlskel       . get sender bank offset
                                       .   (parameter 1)
           la        a2,x10            . get rcvr bank offset (ALS)
           lxsi,u    a1,1              . set sndr increment (short)
           lxsi,u    a2,1              . set rcvr increment (short)
           bt        a2,b1,*a1,b0,*0   . move parlist to ALS
           sbu       b0,a1             . save code bank L,BDI in A1
           aa,u      a1,ccsnum         . create VA of ccs number
           sa        a1,*1,x10,b1      . save VA in parlist
           sbu       b0,a1             . save code bank L,BDI in A1
           aa,u      a1,ccsnam         . create VA of ccs name
           sa        a1,*4,x10,b1      . save VA in parameter 2
           sbu       b1,a1             . save L,BDI of ALS in A1
```

```
            aa,u      a1,x10            . create parlist VA for SCS
    .
    . Clear Register A2 and A3 for the standard calling sequence.
    .
            dsl       a2,72             . clear registers A2, A3
            la,u      a0,3              . number of parameters
            lx,u      x2,cnb2cnmva      . get entry point
            call      0,x2,b0           . xfer control to cnb2cnm
            la,u      a5,0,a0           . get error code only
            jnz       a5,processerr     . jump to process error
    .
    . CCS_NUM_TO_CCS_NAME was successful.
    .
    processerr                          . error processing
    .
    . Error processing code goes here.
    .
            j         comrtn            . jump to common return
    comrtn                              .
            ax,u      x10,als_size      . sell ALS frame
            rtn                         . return to caller
            $end                        . end ccsnum2ccsnam
```

6.7. CCS_TO_CCS Service Routine

Entry Points

EMSM$CCS2CCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

Calling Sequence

```
ccs2ccsva +       EMSM$CCS2CCS      . VA of CCS_TO_CCS
            LA,U    A0,3              . three parameters in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,ccs2ccsva      . load address of service routine
            CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following are the CCS_TO_CCS parameters and the required input for each:

conversion-descriptor

is the internal identifier for the transliteration tables. These tables are returned by OPEN_CCS_TO_CCS.

Parameter Type: Immediate Data

from-string

is the string containing characters to be transliterated.

Parameter Type: Ordinary Data
Required Access: Read

to-string

is the buffer into which CCS_TO_CCS places the transliterated string. If the error status is not OK, this value is undefined.

Parameter Type: Ordinary Data
Required Access: Read and Write

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the bit length of the transliterated string indicated by to-string.

Error Statuses

See 6.30 for the error statuses.

Sample Code

See 6.30 for sample code.

6.8. CLOSE_CCS_TO_CCS Service Routine

Entry Points

EMSM$CLOSCCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

Calling Sequence

```
closccsva +       EMSM$CLOSCCS      . VA of CLOSE_CCS_TO_CCS
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,closccsva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the CLOSE_CCS_TO_CCS parameter and the required input:

conversion-descriptor

is the internal identifier for the transliteration tables. These tables are returned by OPEN_CCS_TO_CCS.

Parameter Type: Immediate Data

Return Values)

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Error Statuses

The string transliteration service routines can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 4 – ISLFRCCSNDEF
- 5 – ISLTOCCSNDEF
- 13 – ISLBADCHAR
- 14 – ISLBADDESC
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 17 – ISLOVERLAPOP
- 30 – ISLCORRUPTAB
- 31 – ISLCCSNOTFOU
- 32 – ISLFROMEQTO

Sample Code

This code example shows how to use OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS to transliterate a character string from one CCS to another. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
```

```
          $rel                        . indicate EM relocatable
          $ascii                      . ASCII characters
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
openccsva +         EMSM$OPENCCS      . VA of open_ccs
ccs2ccsva +         EMSM$CCS2CCS      . VA of ccs_to_ccs
closccsva +         EMSM$CLOSCCS      . VA of close_ccs
from_ccs  $equ      19                . ISO 646fr is from_ccs
to_ccs    $equ      35                . ISO 8859-1 is to_ccs
cv_des    $res      1                 . conversion descriptor
f_string  'résumé'                    . define from_string
fstrlen   $equ      $sl('résumé')     . char. length of string
outbuf    $res      2                 . 2-word limit for output buffer
outlen    $equ      2                 . output length in 2 words
.
. Skeleton of parameter list passed to EMSM$OPENCCS:
.   Parameter 1:  from_CCS, immediate data
.   Parameter 2:  to_CCS, immediate data
.
plskelopen                            .
          pardes    2,0,0,0,0,0,0,0   . parameter 1 descriptor
          +         from_ccs          . from_ccs number
          +         0                 . unused for a 1-word value
          pardes    2,0,0,0,0,0,0,0   . parameter 2 descriptor
          +         to_ccs            . to_ccs number
          +         0                 . unused for a 1-word value
parlstlen1 $equ     $-plskelopen      . par. list len. for openccs
als_size1 $equ      parlstlen1+1      . ALS size for openccs;
                                      . 1 extra word for cv-des
.
. Skeleton of parameter list passed to EMSM$CCS2CCS:
.   Parameter 1:  immediate data (ID)
.   Parameter 2:  f_string, ordinary data (OD), read access
.   Parameter 3:  outbuf, OD, read, write access
.
plskelccs                             .
          pardes    2,0,0,0,0,0,0,0   . parameter 1 descriptor
          +         0                 . value of cv_des
          +         0                 . unused for a 1-word value
          pardes    0,0,1,0,0,1,0,0   . parameter 2 descriptor
          +         0                 . VA of f_string
          +         9*fstrlen         . bit length of f_string
          pardes    0,0,1,0,0,1,1,0   . parameter 3 descriptor
          +         0                 . VA of output buffer
          +         outlen*36         . bit length of outbuf
parlstlen2 $equ     $-plskelccs       . length of parameter. list
                                      .   for ccs_to_ccs
als_size2  $equ     parlstlen2+outlen . ALS size for ccs2ccs
.
. Skeleton of parameter list passed to EMSM$CLOSCCS:
.   Parameter 1:  cv_des, ID
.
plskelclose                           .
```

```
          pardes    2,0,0,0,0,0,0,0   . parameter 1 descriptor
          +         0                 . value of cv_des
          +         0                 . unused for a 1-word value
parlstlen3 $equ     $-plskelclose     . length of parameter list
als_size3 $equ      parlstlen3        . ALS size for close_ccs2ccs
.
$(1)                                  .
.
. Call OPEN_CCS_TO_CCS to obtain conversion descriptor.
.
callccs2ccs*                          .
          buy       *als_size1,x10,b1 . buy ALS frame
          lr,u      r1,parlstlen1     . get len. of param. list 1
          la,u      a1,plskelopen     . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set sender increment
          lxsi,u    a2,1              . set receiver increment
          bt        a2,b1,*a1,b0,*0   . move param. list to ALS
          sbu       b1,a1             . save L,BDI of ALS in A1
          aa,u      a1,x10            . create par. list VA for SCS
          la,u      a0,2              . # of parameters (2)
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          lx,u      x2,openccsva      . get entry point
          call      0,x2,b0           . xfer control to openccs
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. OPEN_CCS_TO_CCS was successful.
. Register A0 contains error status.  It should be zero.
. Register A1 contains conversion descriptor.
.
          la,u      a2,cv_des         . get addr. of cv_des
          sa        a1,0,a2,b0        . save conversion desc.
          ax,u      x10,als_size1     . sell ALS frame
.
. Call CCS_TO_CCS to transliterate the from_string.
.
          buy       *als_size2,x10,b1 . buy ALS frame
          lr,u      r1,parlstlen2     . get len. of par. list 2
          la,u      a1,plskelccs      . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set sender increment
          lxsi,u    a2,1              . set receiver increment
          bt        a2,b1,*a1,b0,*0   . move par. list to ALS
          la,u      a2,cv_des         . get addr. of cv_des
          la        a1,0,a2,b0        . load conversion descriptor
          sa        a1,*1,x10,b1      . save cv_des in ALS
          sbu       b0,a1             . save L,BDI of code bank
          aa,u      a1,f_string       . create VA of f_string
```

```
          sa        a1,*4,x10,b1      . save f_string VA in ALS
          sbu       b0,a1             . save code bank L,BDI in A1
          aa,u      a1,outbuf         . create VA of output buffer
          sa        a1,*7,x10,b1      . save outbuf VA in ALS
          sbu       b1,a1             . save L,BDI of ALS in A1
          aa,u      a1,x10            . create par. list VA for SCS
          la,u      a0,3              . # of parameters (3)
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          lx,u      x2,ccs2ccsva      . get entry point
          call      0,x2,b0           . xfer control to ccs2ccs
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. CCS_TO_CCS was successful.  Register A0 should contain 0.
.
          ax,u      x10,als_size2     . sell ALS frame
.
. Call CLOSE_CCS_TO_CCS.
.
          buy       *als_size3,x10,b1 . buy ALS frame
          lr,u      r1,parlstlen3     . get len. of par. list 3
          la,u      a1,plskelclose    . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set sender increment
          lxsi,u    a2,1              . set receiver increment
          bt        a2,b1,*a1,b0,*0   . move par. list to ALS
          la,u      a2,cv_des         . get addr. of cv_des
          la        a1,0,a2,b0        . load conversion descriptor
          sa        a1,*1,x10,b1      . save cv_des in ALS
          sbu       b1,a1             . save L,BDI of ALS in A1
          aa,u      a1,x10            . create par. list VA for SCS
          la,u      a0,1              . # of parameters (1)
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          lx,u      x2,closccsva      . get entry point
          call      0,x2,b0           . xfer control to closccs
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. CLOSE_CCS_TO_CCS was successful.
. Register A0 contains error status.  It should be zero.
.
          j         comrtn            . jump to common return
processerr                          . error processing
          j         comrtn            . jump to common return
comrtn                              .
          ax,u      x10,als_size3     . sell ALS frame
```

```
                rtn                         . return to caller
                $end                        . end string transliterate
```

6.9. CONVERT_FORM Service Routine

Entry Point

EMSM$CNVRTFM

Description

CONVERT_FORM performs these tasks:

- Converts a character from its current representation in one portion of a character set to its representation in another portion of the same character set. There is always a one-to-one correspondence in the transliteration that occurs.
- Uses the current value of the locale in the LC_CTYPE category to determine which transliteration table to use. The calling program must ensure that LC_CTYPE is set to a locale that contains the appropriate transliteration table.

Calling Sequence

```
cnvrtfmva +       EMSM$CNVRTFM      . VA of CONVERT_FORM
             LA,U    A0,3              . three parameters in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,cnvrtfmva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following are the CONVERT_FORM parameters and the required input for these entries:

in-string

is the string containing the characters to be transliterated.

Parameter Type: Ordinary Data
Required Access: Read

out-string

is the buffer for the converted character string. This is where CONVERT_FORM should write the string. The output buffer can be the same as the input buffer; it must be at least as large as the input string.

Parameter Type: Ordinary Data
Required Access: Read and Write

convert-direction

indicates if the characters are to be converted to the 7-bit or the 8-bit range.

0 = convert to the 7-bit range.

1 = convert to the 8-bit range.

Parameter Type: Immediate Data

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Error Statuses

CONVERT_FORM can return the following error statuses in Register A0:

- 0 – ISLOK
- 3-ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 17 – ISLOVERLAPOP
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 27 – ISLNOTINTER
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use CONVERT_FORM to change the encoding of a character string. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
  $(1)                                 .
           $lit                        . literals to loc. counter1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
  pardes   $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
```

```
cnvrtfmva +         EMSM$CNVRTFM      . VA of convert_form
instring  'süß'                       . character string
instrlen  $equ      $sl('süß')        . length of instring
outlen    $equ      5                 . limit to 5 words
.
. Skeleton of parameter list passed to EMSM$CNVRTFM:
.   Parameter 1:  instring, ordinary data (OD), read access
.   Parameter 2:  outstring, OD, read, write access
.   Parameter 3:  direction, immediate data
.
parlskel                              .
          pardes    0,0,1,0,0,1,1,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         9*instrlen        . bit length of instring
          pardes    0,0,1,0,0,1,1,0   . parameter 2 descriptor
          +         0                 . VA of string
          +         36*outlen         . bit length of output string
          pardes    2,0,0,0,0,0,0,0   . parameter 3 descriptor
          +         0                 . immediate value of 0
          +         1                 . transliterate from 7-bit
                                      .   to 8-bit range
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen+outlen  . ALS frame size
.
. Code begins here.
.
callcnvrtfm*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,instring       . create VA of string
          sa        a2,*1,x10,b1      . set string VA in par. list
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of par. list
          au,u      a1,parlstlen      . create VA of outstring
          sa        a2,*4,x10,b1      . save VA in par. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,3              . number of parameters
          lx,u      x2,cnvrtfmva      . load VA of convert_form
          call      0,x2,b0           . call convert_form
          la,u      a5,0,a0           . get error code only
```

```
          jnz       a5,processerr     . jump to process error
.
. CONVERT_FORM was successful.  The transliterated string is
. located at *parlstlen,x10,b1.
.
subexit                               .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. CONVERT_FORM returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                            . error processing
          tn        a0                . error or warning?
          j         processwarn       . warning, process it
                                      . error-convert_form failed
.
. Error processing goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                           .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end convert_form
```

6.10. GET_COLLATE_TABLE Service Routine

Entry Point

EMSM$GCOLTAB

Description

GET_COLLATE_TABLE is a MASM-callable service routine that copies the collation tables from the I18NLIB common bank into the application's memory space. These tables are indicated by the current setting of the LC_COLLATE category. This operation

- Allows faster processing for the STRING_COLLATE and STRING_TRANSFORM service routines.
- Enables these service routines to access the collating rules directly from the application data space rather than having to establish a pointer to the I18NLIB AFCB.

Calling Sequence

```
gcoltabva +       EMSM$GCOLTAB      . VA of GET_COLLATE_TABLE
             LA,U    A0,n              . one or two parameters in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,gcoltabva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following are the GET_COLLATE_TABLE parameters and the required input for each:

locale

is the name of the locale. If the virtual address for this parameter is zero (0), GET_COLLATE_TABLE uses the locale identified by the LC_COLLATE category.

Parameter Type: Ordinary Data
Required Access: Read

collate-table

is the buffer into which the collate table is to be written.

- If the virtual address for *locale* is zero, this table is for the locale in force.
- If the virtual address for *locale* is non-zero, this table is for the requested locale.

Parameter Type: Ordinary Data
Required Access: Read and Write

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

- If one parameter is passed in the parameter list, Register A1 contains the number of words required to save the LC_COLLATE table for the requested locale or the locale in force.
- If two parameters are passed in the parameter list, Register A1 contains the number of words transferred to the buffer area.
- If a Buffer Too Short error status was returned, Register A1 contains the number of words required to save the collate table for the requested locale or the locale in force.

Error Statuses

GET_COLLATE_TABLE can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use GET_COLLATE_TABLE to copy the collation table for the locale indicated by LC_COLLATE into the buffer col_table. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
$(1)                                   .
           $lit                        . literals to loc. counter 1
bufsize    $equ      500               . buffer size in words
col_table  $res      bufsize           . # words for xform buffer
gcoltblva  +         EMSM$GCOLTBL      . VA of get_collate_table
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
.
. Skeleton of first parameter list passed to EMSM$GCOLTBL:
.   Parameter 1: locale, ordinary data (OD), read access
.   Parameter 2: collate table, OD, read, write access
.
parlskel1                              .
           pardes    0,0,0,0,0,1,1,0   . parameter 1 descriptor
           +         0                 . VA of locale
           +         0                 . # of char. in locale
parlskel2                              .
           pardes    0,0,0,0,0,1,1,0   . parameter 2 descriptor
           +         0                 . VA of buffer
           +         bufsize*36        . bit length of buffer
```

```
parlstlen1 $equ     $-parlskel1       . length of param. list 1
als_size1 $equ      parlstlen1+bufsize . ALS size
.
callgcolex*                           .
          buy       *als_size1,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen1     . get len. of par. list 1
          la,u      a1,parlskel1      . get sender bank offset
                                      .   of par. list 2
          la        a2,x10            . get receiver bank offset ALS
          lxsi,u    a1,1              . set sndr increment (short)
          lxsi,u    a2,1              . set rcvr increment (short)
          bt        a2,b1,*a1,b0,*0   . move par. list to ALS
          sbu       b0,a1             . get code bank L,BDI;
                                      .   save in A1
          aa,u      a1,col_table      . create VA of collate table
          sa        a1,*4,*x10,b1     . save str1 VA in par. list
          sbu       b1,a1             . save L,BDI of ALS in A1
          aa,u      a1,x10            . create par. list VA for SCS
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,gcoltblva      . get entry point
          call      0,x2,b0           . xfer control to gcoltbl
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. GET_COLLATE_TABLE was successful.
.
subexit                               .
          ax,u      x10,als_size1     . sell ALS frame
          rtn                         . return to caller
.
. A0 still has the error information.
.
Processerr                            . error processing
          j         subexit           . jump to common exit
          $end                        . end get_collate_table
```

6.11. GETENV Service Routine

Entry Point

EMSM$GETENV

Description

GETENV retrieves the current value of the environment variable set at the run level. These variables are initially set by the default values from the user profile.

Calling Sequence

```
getenvva +        EMSM$GETENV       . virtual address of GETENV
            LA,U     A0,2              . two parameters in list
            LA       A1,virtual-address-of-parameter-list
            LX,U     X2,getenvva       . load address of service routine
            CALL     0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, each parameter format is a three-word entry. The following are the GETENVSYS parameters and the required input for these entries:

environment-variable-name

is the name of the environment variable for which the value is being requested.

Parameter Type: Ordinary Data
Required Access: Read

environment-variable-value

is the buffer for the environment variable value. This is where GETENVSYS should write the value of *environment-variable-name*.

Parameter Type: Ordinary Data
Required Access: Read and Write

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

environment-variable-value

contains the value for the specified *environment-variable-name*. If the error status in Register A0 is not OK, this value is undefined.

Register A1

contains the bit length of *environment-variable-value*. If the GETENVSYS operation fails, the value in Register A1 is undefined.

Error Statuses

GETENV can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR

- 20 – ISLNAMINVCHA
- 21 – ISLVALINVCHR
- 24 – ISLNAMNOTFOU
- 39 – ISLGETENVERR

Sample Code

GETENV is used to retrieve the value of environment variables (EV) at the run level. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                  .
          $lit                        . literals to loc. counter 1
          $ascii                      . literals in ASCII
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
getenvva  +         EMSM$GETENV       . virtual addr.of getenv
evnam     'LC_CTYPE'                  .
evnamlen  $equ      $-evnam           . word length of EV name
evvallen  $equ      12                . 12-word limit to EV value
.
. Skeleton of parameter list passed to EMSM$GETENV:
.   Parameter 1:  EV name, ordinary data (OD), read access
.   Parameter 2:  EV value, OD, read, write access
.
parlskel                              .
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . virtual addr. of EV name
          +         36*evnamlen       . length of name in bits
          pardes    0,0,0,0,0,1,1,0   . parameter 2 descriptor
          +         0                 . virtual addr. of value
          +         evvallen          . EV value buffer word len.
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen+evvallen . ALS frame size
.
. Code begins here.
.
callgetenv*                           .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . param. list len. to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set the increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
```

```
          aa,u      a2,evnam          . create VA of EV name
          sa        a2,*1,x10,b1      . set EV_NAM VA in param. list
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of param. list
          au,u      a1,parlstlen      . create VA of EV_VALUE
          sa        a2,*4,x10,b1      . save VA in param. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,2              . number of parameters
          lx,u      x2,getenvva       . load VA of getenv
          call      0,x2,b0           . call getenv
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. GETENV was successful.  A1 has length of EV_VALUE 4,x10;
. B1 has EV_VALUE.
.
. Processing of value code goes here.
.
subexit                             .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. GETENV returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                          . error processing
          tn        a0                . error or warning?
          j         processwarn       . warning, process it
                                      . error-getenv failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                         .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end getenv
```

6.12. GETENVSYS Service Routine

Entry Point

EMSM$GETENVS

Description

GETENVSYS retrieves the current value of the environment variable set at the system-level.

Calling Sequence

```
getenvsva +       EMSM$GETENVS      . virtual addr. of GETENVSYS
            LA,U    A0,2              . two parameters in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,getenvsva      . load address of service routine
            CALL 0,X2,B0              . call service routine
```

Parameter List

See 6.11 for the parameter list.

Return Values

See 6.11 for the return values.

Error Statuses

GETENVSYS can return the following error statuses in Register A0:

- 0-ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHA
- 21 – ISLVALINVCHR
- 24 – ISLNAMNOTFOU
- 39 – ISLGETENVERR

Sample Code

GETENVSYS is used to retrieve the value of environment variables (EV) at the system level. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
  (1)                                  .
           $lit                        . literals to loc. counter 1
           $ascii                      . literals in ASCII
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
  pardes   $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
  getenvsva +        EMSM$GETENVS      . VA of getenvsys
  evnam    'LC_CTYPE'                  .
```

```
evnamlen  $equ      $-evnam           . word len. of EV name
.
. Skeleton of parameter list passed to EMSM$GETENVS:
.   Parameter 1:  EV name, ordinary data (OD), read access
.   Parameter 2:  EV value, OD, read, write access
.
parlskel                              .
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of EV name
          +         36*evnamlen       . length of name in bits
          pardes    0,0,0,0,0,1,1,0   . parameter 2 descriptor
          +         0                 . VA of EV value
          +         evvallen          . EV value buffer word length
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
evvallen  $equ      12                . 12-word limit to EV value
als_size  $equ      parlstlen+evvallen . ALS frame size
.
. Code begins here.
.
callgetenvs*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . param. list len. to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set the increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,evnam          . create VA of EV name
          sa        a2,*1,x10,b1      . set EV_NAM VA in par. list
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of param. list
          au,u      a1,parlstlen      . create VA of EV_VALUE
          sa        a2,*4,x10,b1      . save VA in parameter list
.
. Clear Registers A2 And A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,2              . number of parameters
          lx,u      x2,getenvsva      . load VA of getenvsys
          call      0,x2,b0           . call getenvsys
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. GETENVSYS was successful.  A1 has length of EV_VALUE 4,x10,b1
. has EV_VALUE.
.
. Processing of value code goes here.
.
```

```
subexit                                 .
          ax,u      x10,als_size        . sell ALS frame
          rtn                           . return to caller
.
. GETENVSYS returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                              . error processing
          tn        a0                  . error or warning?
          j         processwarn         . warning, process it
                                        . error-getenvsys failed
.
. Error processing goes here.
.
          j         subexit             . exit through common point
.
. Process the warning.
.
processwarn                             .
.
. Warning processing code goes here.
.
          j         subexit             . exit through common point
          $end                          . end getenvsys
```

6.13. ISALNUM Service Routine

Entry Point

EMSM$ISALNUM

Description

ISALNUM determines if all the characters in the string are in the alphanumeric character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *alpha* and *digit* categories in the locale definition file.

Calling Sequence

```
isalnumva +       EMSM$ISALNUM      . virtual addr. of ISALNUM
            LA,U    A0,1              . one parameter in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,isalnumva      . load address of service routine
            CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISALNUM to determine if all characters of the input string are alphanumeric. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
isalnumva +          EMSM$ISALNUM      . VA of isalnum
char_string 'String to test'           . character string to test
.
. Skeleton of parameter list passed to EMSM$ISALNUM:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ       $-parlskel        . length of parameter list
als_size  $equ       parlstlen         . ALS frame size
.
. Code begins here.
.
callisalnum*                           .
           buy       *als_size,*x10,b1 . buy ALS frame
           lr,u      r1,parlstlen      . par. list length to move
           la,u      a1,parlskel       . get sender bank offset
           la        a2,x10            . get receiver bank offset
           lxsi,u    a1,1              . set increments
           lxsi,u    a2,1              .
           bt        a2,b1,*a1,b0,*0   . move parameter list
           sbu       b0,a2             . get L,BDI of code bank
           aa,u      a2,char_string    . create VA of string
           sa        a2,*1,x10,b1      . set string VA in par. list
           sbu       b1,a1             . get L,BDI of ALS
           aa        a1,x10            . create VA of par. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
           dsl       a2,72             . clear registers A2, A3
           la,u      a0,1              . number of parameters
           lx,u      x2,isalnumva      . load isalnum address
           call      0,x2,b0           . call isalnum
```

```
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISALNUM was successful; all characters are alphanumeric.
.
subexit                               .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISALNUM returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                            . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning, process it
                                      . error-isalnum failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                           .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end isalnum
```

6.14. ISALPHA Service Routine

Entry Point

EMSM$ISALPHA

Description

ISALPHA determines if all the characters in the string are in the alphabetic character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *upper*, *lower*, and *letter* categories in the locale definition file.

Calling Sequence

```
isalphava +       EMSM$ISALPHA      . virtual addr. of ISALPHA
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,isalphava      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB

- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISALPHA to determine if all characters of the input string are alphabetic. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
isalphava +          EMSM$ISALPHA      . VA of isalpha
char_string 'String to test'           . character string to test
.
. Skeleton of parameter list passed to EMSM$ISALPHA:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ       $-parlskel        . length of parameter list
als_size  $equ       parlstlen         . ALS frame size
.
. Code begins here.
.
CALLISALPHA*                           .
           buy       *als_size,*x10,b1 . buy ALS frame
           lr,u      r1,parlstlen      . par. list length to move
           la,u      a1,parlskel       . get sender bank offset
           la        a2,x10            . get receiver bank offset
           lxsi,u    a1,1              . set increments
           lxsi,u    a2,1              .
           bt        a2,b1,*a1,b0,*0   . move parameter list
           sbu       b0,a2             . get L,BDI of code bank
           aa,u      a2,char_string    . create VA of string
           sa        a2,*1,x10,b1      . set string VA in par. list
           sbu       b1,a1             . get L,BDI of ALS
           aa        a1,x10            . create VA of par. list
```

```
.
. Clear Registers A2 And A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,isalphava      . load isalpha address
          call      0,x2,b0           . call isalpha
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
.  ISALPHA was successful; all characters are alphabetic.
.
subexit                             .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISALPHA returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                          . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-isalpha failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                         .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end isalpha
```

6.15. ISCNTRL Service Routine

Entry Point

EMSM$ISCNTRL

Description

ISCNTRL determines if all the characters in the string are in the control character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *cntrl* category in locale definition file.

Calling Sequence

```
iscntrlva +       EMSM$ISCNTRL      . virtual addr. of ISCNTRL
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,iscntrlva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB

- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISCNTRL to determine if all characters of the input string are control characters. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                  .
          $lit                        . literals to loc. counter 1
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
          $ascii                      . ASCII characters
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
iscntrlva +         EMSM$ISCNTRL      . VA of iscntrl
char_string 'String to test'          . character string to test
.
. Skeleton of parameter list passed to EMSM$ISCNTRL:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                              .
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
. Code begins here.
.
calliscntrl*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,char_string    . create VA of string
          sa        a2,*1,x10,b1      . set string VA in par. list
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of par. list
```

```
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,iscntrlva      . load iscntrl address
          call      0,x2,b0           . call iscntrl
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISCNTRL was successful; all characters are control characters.
.
subexit                               .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISCNTRL returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                            . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-iscntrl failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                           .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end iscntrl
```

6.16. ISDIGIT Service Routine

Entry Point

EMSM$ISDIGIT

Description

ISDIGIT determines if all the characters in the string are decimal digits in the numeric character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *digit* category in the locale definition file.

Calling Sequence

```
isdigitva +       EMSM$ISDIGIT      . virtual addr. of ISDIGIT
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,isdigitva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR

- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISDIGIT to determine if all the characters of the input string are numeric (decimal digits). In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                    .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
isdigitva +          EMSM$ISDIGIT      . VA of isdigit
char_string 'String to test'           . character string to test
.
. Skeleton of parameter list passed to EMSM$ISDIGIT:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ       $-parlskel        . length of parameter list
als_size  $equ       parlstlen         . ALS frame size
.
. Code begins here.
.
callisdigit*                           .
           buy       *als_size,*x10,b1 . buy ALS frame
           lr,u      r1,parlstlen      . par. list length to move
           la,u      a1,parlskel       . get sender bank offset
           la        a2,x10            . get receiver bank offset
           lxsi,u    a1,1              . set increments
           lxsi,u    a2,1              .
           bt        a2,b1,*a1,b0,*0   . move parameter list
           sbu       b0,a2             . get L,BDI of code bank
           aa,u      a2,char_string    . create VA of string
```

```
          sa        a2,*1,x10,b1      . set str. VA in par. list
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of par. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,isdigitva      . load isdigit address
          call      0,x2,b0           . call isdigit
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISDIGIT was successful; all characters are digits.
.
subexit                             .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISDIGIT returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                          . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-isdigit failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                         .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end isdigit
```

6.17. ISGRAPH Service Routine

Entry Point

EMSM$ISGRAPH

Description

ISGRAPH determines if all the characters in the string are in the graphic character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *graph* category in the locale definition file.

Calling Sequence

```
isgraphva +       EMSM$ISGRAPH      . virtual addr. of ISGRAPH
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,isgraphva      . load address of service routine
             CALL      0,X2,B0         . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR

- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISGRAPH to determine if all the characters of the input string are graphic characters. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                  .
          $lit                        . literals to loc. counter 1
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
          $ascii                      . ASCII characters
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
isgraphva +         EMSM$ISGRAPH      . VA of isgraph
char_string 'String to test'          . character string to test
.
. Skeleton of parameter list passed to EMSM$ISGRAPH:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                              .
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
. Code begins here.
.
callisgraph*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,char_string    . create VA of string
```

```
          sa        a2,*1,x10,b1      . set string VA in parlist
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of parlist
.
. Clear Register A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,isgraphva      . load isgraph address
          call      0,x2,b0           . call isgraph
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISGRAPH was successful; all characters are graphical.
.
subexit                             .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISGRAPH returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                          . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-isgraph failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
Processwarn                         .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end isgraph
```

6.18. ISLOWER Service Routine

Entry Point

EMSM$ISLOWER

Description

ISLOWER determines if all the characters in the string are in the lowercase character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *lower* category in the locale definition file.

Calling Sequence

```
islowerva +       EMSM$ISLOWER      . virtual addr. of ISLOWER
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,islowerva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR

- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISLOWER to determine if the characters of the input string are lowercase and alphabetic. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                  .
          $lit                        . literals to loc. counter 1
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
          $ascii                      . ASCII characters
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
islowerva +         EMSM$ISLOWER      . VA of islower
char_string 'String to test'          . character string to test
.
. Skeleton of parameter list passed to EMSM$ISLOWER:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
. Code begins here.
.
callislower*
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par.list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    1,1               . set increments
          lxsi,u    a2,1              .
```

```
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,char_string    . create VA of string
          sa        a2,*1,x10,b1      . set string VA in parlist
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of par. list
.
. Clear Register A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,islowerva      . load islower address
          call      0,x2,b0           . call islower
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISLOWER was successful; all characters are lowercase.
.
subexit                             .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISLOWER returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                          . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-islower failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                         .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end islower
```

6.19. ISPRINT Service Routine

Entry Point

EMSM$ISPRINT

Description

ISPRINT determines if all the characters in the string are in the printable character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *print* category in the locale definition file.

Calling Sequence

```
isprintva +       EMSM$ISPRINT      . virtual addr. of ISPRINT
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,isprintva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR

- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISPRINT to determine if the characters of the input string are printable. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                  .
          $lit                        . literals to loc. counter 1
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
          $ascii                      . ASCII characters
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
isprintva +         EMSM$ISPRINT      . VA of isprint
char_string 'String to test'          . character string to test
.
. Skeleton of parameter list passed to EMSM$ISPRINT:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                              .
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
. Code begins here.
.
callisprint*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
```

```
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,char_string    . create VA of string
          sa        a2,*1,x10,b1      . set str. VA in parlist
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of parlist
.
. Clear Register A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,isprintva      . load isprint address
          call      0,x2,b0           . call isprint
          la,u     a5,0,a0            . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISPRINT was successful; all characters are printable.
.
subexit                              .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISPRINT returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                           . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-isprint failed
.
.  Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end isprint
```

6.20. ISPUNCT Service Routine

Entry Point

EMSM$ISPUNCT

Description

ISPUNCT determines if all the characters in the string are in the punctuation character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *punct* category in the locale definition file.

Calling Sequence

```
ispunctva +       EMSM$ISPUNCT      . virtual addr. of ISPUNCT
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,ispunctva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines:

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISPUNCT to determine if the characters of the input string are punctuation characters. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
ispunctva +          EMSM$ISPUNCT      . VA of ispunct
char_string 'String to test'           . character string to test
.
. Skeleton of parameter list passed to EMSM$ISPUNCT:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ       $-parlskel        . length of parameter list
als_size  $equ       parlstlen         . ALS frame size
.
```

```
. Code begins here.
.
callispunct*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,char_string    . create VA of string
          sa        a2,*1,x10,b1      . set str VA in parlist
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of parlist
.
. Clear Register A2 And A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,ispunctva      . load ispunct address
          call      0,x2,b0           . call ispunct
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISPUNCT successful; all characters are punctuation characters.
.
subexit                               .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISPUNCT returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                            . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-ispunct failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                           .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end ispunct
```

6.21. ISSPACE Service Routine

Entry Point

EMSM$ISSPACE

Description

ISSPACE determines if all the characters in the string are in the white space character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *space* category in the locale definition file.

Calling Sequence

```
isspaceva +       EMSM$ISSPACE      . virtual addr. of ISSPACE
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,isspaceva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISSPACE to determine if the characters of the input string are white space characters. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
isspaceva +          EMSM$ISSPACE      . VA of isspace
char_string 'String to test'           . character string to test
.
. Skeleton of parameter list passed to EMSM$ISSPACE:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         9*14              . length of name in bits
.
. Define length parameters.
.
```

```
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
. Code begins here.
.
callisspace*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . parlist length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,char_string    . create VA of string
          sa        a2,*1,x10,b1      . set str VA in parlist
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of parlist
.
. Clear Register A2 And A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,isspaceva      . load isspace address
          call      0,x2,b0           . call isspace
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISSPACE successful; all characters are space characters.
.
subexit                               .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISSPACE returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                            . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-isspace failed
.
.  Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                           .
.
. Warning processing code goes here.
.
```

```
            j         subexit           . exit through common point
            $end                        . end isspace
```

6.22. ISUPPER Service Routine

Entry Point

EMSM$ISUPPER

Description

ISUPPER determines if all the characters in the string are in the uppercase character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the *upper* category in the locale definition file.

Calling Sequence

```
isupperva +       EMSM$ISUPPER      . virtual addr. of ISUPPER
            LA,U    A0,1              . one parameter in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,isupperva      . load address of service routine
            CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISUPPER to determine if the characters of the input string are uppercase and alphabetic. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
isupperva +          EMSM$ISUPPER      . VA of isupper
char_string 'String to test'           . character string to test
.
. Skeleton of parameter list passed to EMSM$ISUPPER:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         9*14              . length of name in bits
.
```

```
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
. Code begins here.
.
callisupper*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,char_string    . create VA of string
          sa        a2,*1,x10,b1      . set str VA in parlist
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of parlist
.
. Clear Register A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,isupperva      . load isupper address
          call      0,x2,b0           . call isupper
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISUPPER was successful; all characters are uppercase.
.
subexit                               .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISUPPER returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                            . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-isupper failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                           .
.
```

```
   . Warning processing code goes here.
   .
             j         subexit           . exit through common point
             $end      isupperex         . end isupper
```

6.23. ISXDIGIT Service Routine

Entry Point

EMSM$ISXDIG

Description

ISXDIGIT determines if all the characters in the string are in the hexadecimal digit (xdigit) character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE.

The allowable values for a locale are determined by the setting of the *xdigit* category in the locale definition file.

Calling Sequence

```
isxdigitva +      EMSM$ISXDIG       . virtual addr. of ISXDIGIT
              LA,U   A0,1              . one parameter in list
              LA     A1,virtual-address-of-parameter-list
              LX,U   X2,isxdigitva     . load address of service routine
              ALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. This parameter is used for all the character class service routines.

character-string

is the character string to be tested.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

These registers contain the return values passed by all the character class service routines.

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains one of the following binary numbers:

1 = if the characters are the correct character class for the service routine.

0 = if one or more characters are not the correct character class, or an error occurred. If Register A1 is 0, check Register A0 to see if an error occurred.

Error Statuses

These are the error statuses that can be returned in Register A0 by the character class service routines

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use ISXDIGIT to determine if the characters of the input string are hexadecimal digits. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
isxdigitva +         EMSM$ISXDIGIT     . VA of isxdigit
char_string 'String to test'           . character string to test
.
. Skeleton of parameter list passed to EMSM$ISXDIGIT:
.   Parameter 1:  char_string, ordinary data (OD), read access
.
parlskel                               .
```

```
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         9*14              . length of name in bits
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
. Code begins here.
.
callisxdigit*                         .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,char_string    . create VA of string
          sa        a2,*1,x10,b1      . set str. VA in parlist
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of parlist
.
. Clear Register A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,isxdigitva     . load isxdigit address
          call      0,x2,b0           . call isxdigit
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. ISXDIGIT was successful; all characters are alphanumeric.
.
subexit                               .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. ISXDIGIT returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                            . error processing
          tn        a5                . error or warning?
          j         processwarn       . warning; process it
                                      . error-isdigit failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
```

```
. Process the warning.
.
processwarn                          .
.
. Warning processing code goes here.
.
          j         subexit          . exit through common point
          $end                       . end isxdigit
```

6.24. LOCALE_NAME_TO_CCS_NUM Service Routine

Entry Point

EMSM$LNM2CNB

Description

LOCALE_NAME_TO_CCS_NUM returns the coded character set (CCS) number associated with the locale name.

Calling Sequence

```
lnm2cnbva +       EMSM$LNM2CNB      . VA of LOC_NAM_TO_CCS_NUM
             LA,U    A0,2              . two parameters in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,lnm2cnbva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the LOCALE_NAME_TO_CCS_NUM parameters and the required input:

locale-name

is the string containing the locale name. A locale name is required for the SETLOCALE service routine. This name is case-sensitive and must correspond to (1) a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and (2) a locale that is active on the system.

Parameter Type: Ordinary Data
Required Access: Read

ccs-number-format

is the kind of CCS number desired. The possible values are

0 = the Unisys CCS number used with the CCS-TO-CCS transliteration.

1 = the ISO CCS number for interoperability use.

2 = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records.

Parameter Type: Immediate Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the CCS number corresponding to the specified locale name. If the service routine returns an error status other than OK, the value for Register A1 is undefined.

Error Statuses

LOCALE_NAME_TO_CCS_NUM can return the following error statuses in Register A0:

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 26 – ISLLOCNOTFOU
- 45 – ISLCNUMFORUN

Sample Code

This code example show how to use LOCALE_NAME_TO_CCS_NUM to return the CCS number associated with the locale name. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
$(1)                                   .
           $lit                        . literals to loc. counter 1
lnm2cnbva +         EMSM$LNM2CNB       . VA of locnam2cnum
locnam    'fr_FR.8859-1'               . locale name
locnamlen $equ      $SL('fr_FR.8859-1') . char. len. of locname
pardes    $form     2,1,1,23,1,1,1,6   . IP,L,U,S & Res.(0),E,R,W,B
.
. Skeleton of parameter list passed to EMSM$LNM2CNB:
.   Parameter 1: locnam, ordinary data (OD), read access
.
parlskel
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         9*locnamlen       . bit length of string
```

```
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS size
.
calllnm2cnb*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . get length of parlist
          la,u      a1,parlskel       . get sender bank offset
                                      .  (parameter 1)
          la        a2,x10            . get rcvr bank offset (ALS)
          lxsi,u    a1,1              . set sndr increment (short)
          lxsi,u    a2,1              . set rcvr increment (short)
          bt        a2,b1,*a1,b0,*0   . move parlist to ALS
          sbu       b0,a1             . save code bank L,BDI in A1
          aa,u      a1,locnam         . create VA of locale name
          sa        a1,*1,x10,b1      . save VA in parlist
          sbu       b1,a1             . save L,BDI of ALS in A1
          aa,u      a1,x10            . create parlist VA for SCS
.
. Clear Register A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,lnm2cnbva      . get entry point
          call      0,x2,b0           . xfer control to lnm2cnb
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. LOCALE_NAME_TO_CCS_NUM was successful.
.
processerr                            . error processing
.
. Error processing code goes here.
.
          j         comrtn            . jump to common return
comrtn                                .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
          $end                        . end of locnam2ccsnum
```

6.25. LOCALE_NAME_TO_LOCALE_NUM Service Routine

Entry Point

EMSM$LNM2LNB

Description

LOCALE_NAME_TO_LOCALE_NUM transforms a locale name to its corresponding locale number.

Calling Sequence

```
lnm2lnbva +       EMSM$LNM2LNB      . VA of LOC_NAME_TO_LOC_NUM
             LA,U    A0,1              . one parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,lnm2lnbva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the LOCALE_NAME_TO_LOCALE_NUM parameter and the required input:

locale-name

is the name of the locale. A locale name is required for the SETLOCALE service routine. This name is case-sensitive and must correspond to (1) a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and (2) a locale that is active on the system.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

is the locale number corresponding to the specified name. If the service routine returns an error status other than OK, the value for Register A1 is undefined.

Error Statuses

LOCALE_NAME_TO_LOCALE_NUM can return the following error statuses in Register A0:

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 26 – ISLLOCNOTFOU

Sample Code

This code example shows how to use LOCALE_NAME_TO_LOCALE_NUM to transform a locale name to its corresponding locale number. In this example, it is assumed that MASM produces an extended mode relocatable.

```
         $include  'maxr$/'          . define registers
            $include  'cbep$$isl-em'    . def. I18NLIB entry points
```

```
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
$(1)                                  .
          $lit                        . literals to loc. counter1
lnm2lnbva +         EMSM$LNM2LNB      . VA of locnam2locnum
locnam    'fr_FR.8859-1'              . locale name
locnamlen $equ      $SL('fr_FR.8859-1') . char. length of locnam
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
.
. Skeleton of parameter list passed to LOCALE_NAME_TO_LOCALE_NUM:
.   Parameter 1: locnam, ordinary data (OD), read access
.
parlskel                              .
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         9*locnamlen       . bit length of string
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
calllnm2lnb*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . get length of parlist
          la,u      a1,parlskel       . get sender bank offset
                                      .   (parameter 1)
          la        a2,x10            . get rcvr bank offset (ALS)
          lxsi,u    a1,1              . set sndr increment (short)
          lxsi,u    a2,1              . set rcvr increment (short)
          bt        a2,b1,*a1,b0,*0   . move parlist to ALS
          sbu       b0,a1             . save code bank L,BDI in A1
          aa,u      a1,locnam         . create VA of locale name
          sa        a1,*1,x10,b1      . save VA in parlist
          sbu       b1,a1             . save L,BDI of ALS in A1
          aa,u      a1,x10            . create parlist VA for SCS
.
. Clear Register A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          lx,u      x2,lnm2lnbva      . get entry point
          call      0,x2,b0           . xfer control to lnm2lnb
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. LOCALE_NAME_TO_LOCALE_NUM was successful.
.
processerr                            . error processing
.
. Error processing code goes here.
.
          j         comrtn            . jump to common return
comrtn                                .
          ax,u      x10,als_size      . sell ALS frame
```

```
            rtn                         . return to caller
            $end                        . end locnam2locnum
```

6.26. LOCALE_NUM_TO_CCS_NUM Service Routine

Entry Point

EMSM$LNB2CNB

Description

LOCALE_NUM_TO_CCS_NUM returns the coded character set (CCS) number associated with the locale number.

Calling Sequence

```
lnb2cnbva +       EMSM$LNB2CNB      . VA of LOC_NUM_TO_CCS_NUM
            LA,U    A0,2              . two parameters in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,lnb2cnbva      . load address of service routine
            CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the LOCALE_NUM_TO_CCS_NUM parameters and the required input:

locale-number

is the number of a locale that is active on the system.

Parameter Type: Ordinary Data
Required Access: Read

ccs-number-format

is the kind of CCS number desired. The possible values are

0 = the Unisys CCS number used with the CCS_TO_CCS transliteration.

1 = the ISO CCS number for interoperability use.

2 = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records.

Parameter Type: Immediate Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the CCS number corresponding to the specified name. If the service routine returns an error status other than OK, the value for Register A1 is undefined.

Error Statuses

LOCALE_NUM_TO_CCS_NUM can return the following error statuses in Register A0:

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 26 – ISLLOCNOTFOU
- 45 – ISLCNUMFORUN

Sample Code

This code example shows how to use LOCALE_NUM_TO_CCS_NUM to return the CCS number associated with the locale number. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
$(1)                                   .
           $lit                        . literals to loc. counter 1
lnb2cnbva +         EMSM$LNB2CNB       . VA of lnb2cnb
locnum     $equ     4                  . locale number
pardes     $form    2,1,1,23,1,1,1,6   . IP,L,U,S & Res.(0),E,R,W,B
.
. Skeleton of parameter list passed to EMSM$LNB2CNB:
.   Parameter 1: locnam, ordinary data (OD), read access
.
parlskel                               .
           pardes   0,0,1,0,0,1,0,0    . parameter 1 descriptor
           +        0                  . VA of string
           +        36                 . bit len. of loc. number
           pardes   2,0,0,0,0,0,0,0    . parameter 2 descriptor
           +        2                  . SDF number format
           +        0                  . immediate value of 0
parlstlen  $equ     $-parlskel         . length of parameter list
als_size   $equ     parlstlen          . ALS frame size
.
calllnb2cnb*                           .
```

```
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . get length of parlist
          la,u      a1,parlskel       . get sender bank offset
                                      .   (parameter 1)
          la        a2,x10            . get rcvr bnk offset (ALS)
          lxsi,u    a1,1              . set sndr increment (short)
          lxsi,u    a2,1              . set rcvr increment (short)
          bt        a2,b1,*a1,b0,*0   . move parlist to ALS
          sbu       b0,a1             . save code bank L,BDI in A1
          aa,u      a1,locnum         . locale number
          sa        a1,*1,x10,b1      . save in parameter list
          sbu       b1,a1             . save L,BDI of ALS in A1
          aa,u      a1,x10            . create parlist VA for SCS
.
. Clear Register A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,2              . number of parameters
          lx,u      x2,lnb2cnbva      . get entry point
          call      0,x2,b0           . xfer control to lnb2cnb
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. LOCALE_NUM_TO_CCS_NUM was successful.
.
processerr                          . error processing
.
. Error processing code goes here.
.
          j         comrtn            . jump to common return
comrtn                              .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
          $end                        . end locnum2ccsnum
```

6.27. LOCALE_NUM_TO_LOCALE_NAME Service Routine

Entry Point

EMSM$LNB2LNM

Description

LOCALE_NAME_TO_LOCALE_NUM transforms a locale name to its corresponding locale number.

Calling Sequence

```
lnb2lnmva +       EMSM$LNB2LNM      . VA of LOC_NUM_TO_LOC_NAME
             LA,U    A0,2             . two parameter in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,lnb2lnmva     . load address of service routine
             CALL    0,X2,B0          . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the LOCALE_NUM_TO_LOCALE_NAME parameters and the required input:

locale-number

is a locale number that corresponds to a locale on the system.

Parameter Type: Ordinary Data
Required Access: Read

locale-name

is the buffer into which the service routine should write the corresponding locale name. A locale name is required for the SETLOCALE service routine. This name is case-sensitive and corresponds to (1) a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and (2) a locale that is active on the system. If the service routine returns an error status other than OK, this value is undefined.

Parameter Type: Ordinary Data
Required Access: Read/Write

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the bit length of the locale name. If the service routine returns an error status other than OK, the value for Register A1 is undefined.

Error Statuses

LOCALE_NUM_TO_LOCALE_NAME can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR

- 26 – ISLLOCNOTFOU

Sample Code

This code example shows how to use LOCALE_NAME_TO_LOCALE_NUM to transform a locale name to its corresponding locale number. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
$(1)       $lit                        . literals to loc. counter 1
lnb2lnmva +          EMSM$LNB2LNM      . VA of locnam2locnum
locnam     $res      4                 . locale name
locnamlen $equ       4                 . locale name max. length
locnum     +         6                 . locale number
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
.
. Skeleton of parameter list passed to EMSM$LNB2LNM:
.   Parameter 1: locnum, ordinary data (OD), read access
.   Parameter 2: locnam, OD, read write access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of number
           +         36                . bit length of number
           pardes    0,0,1,0,0,1,1,0   . parameter 2 descriptor
           +         0                 . VA of string
           +         36*locnamlen      . bit length of string
parlstlen $equ       $-parlskel        . length of parameter list
als_size   $equ      parlstlen         . ALS frame size
.
calllnb2lnm*                           .
           buy       *als_size,*x10,b1 . buy ALS frame
           lr,u      r1,parlstlen      . get length of parlist
           la,u      a1,parlskel       . get sender bank offset
                                       .   (parameter 1)
           la        a2,x10            . get rcvr bnk offset (ALS)
           lxsi,u    a1,1              . set sndr increment (short)
           lxsi,u    a2,1              . set rcvr increment (short)
           bt        a2,b1,*a1,b0,*0   . move parlist to ALS
           sbu       b0,a1             . save code bank L,BDI in A1
           aa,u      a1,locnum         . create VA of locale number
           sa        a1,*1,x10,b1      . save VA in parameter list
           sbu       b0,a1             . save code bank L,BDI in A1
           aa,u      a1,locnam         . create VA of locale name
           sa        a1,*4,x10,b1      . save VA in parameter 2
           sbu       b1,a1             . save L,BDI of ALS in A1
           aa,u      a1,x10            . create parlist VA for SCS
.
. Clear Register A2 and A3 for the standard calling sequence.
.
```

```
            dsl       a2,72             . clear registers A2, A3
            la,u      a0,2              . number of parameters
            lx,u      x2,lnb2lnmva      . get entry point
            call      0,x2,b0           . xfer control to lnm2lcnm
            la,u      a5,0,a0           . get error code only
            jnz       a5,processerr     . jump to process error
    .
    . LOCALE_NUM_TO_LOCALE_NAME was successful.
    .
    processerr                          . error processing
    .
    . Error processing code goes here.
    .
            j         comrtn            . jump to common return
    comrtn                              .
            ax,u      x10,als_size      . sell ALS frame
            rtn                         . return to caller
            $end                        . end locnum2locnam
```

6.28. NAME_AND_NUMBER Packet Interface

Entry Point

EMSM$NMNB

Description

The NAME_AND_NUMBER packet interface provides a general interface for obtaining name-and-number information for various locales and CCSs. The Virtual Machine for the Java™ Platform on ClearPath OS 2200 can use this interface to obtain this information, which also includes the number of bits in a character for the requested locale or CCS.

Note: *All locales have an associated CCS number. However, not all CCSs are associated with a locale.*

Calling Sequence

```
nmnbva +          emsm$nmnb          . VA of name-number
          :
          :
          la,u      a0,1               . 1 parameter in list
          la        a1,vapl            . virtual address of parameter list
          lx,u      x2,nmnbva          . load address of service routine
          call      0,x2,b0            . transfer control to service
    routine
```

Error Statuses

NAME_AND_NUMBER can return the following error statuses in Register A0:

- 0 – ISLOK
- 1 – ISLINVPKTLEN
- 2 – ISLINVPKTVER

- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 26 – ISLLOCNOTFOU
- 31 – ISLCCSNOTFOU
- 40 – ISLCNUMNOTUN
- 45 – ISLCNUMFORUN
- 57 – ISLINVNMNBID
- 58-ISLINVNMLEN

See the NAME_AND_NUMBER Packet Interface in the COBOL section for the list of fields in the *name_and_number* information packet.

Sample Code

This code example shows how to use NAME_AND_NUMBER to obtain CCS and locale name-and-number information.

```
        $include  'maxr$/'           . define registers
           $include  'om$def'           . define om definition
           $include  'cbep$$isl-em'     . def. I18NLIB entry points
           $extend                      . indicate extended mode
           $obj                         . indicate EM object modules
           $ascii                       .
OM$CODEG_EMB $EQU  +(OM$BANK_FORM  ;    . CODED ATTRIBUTE FORM
                    OM$CODE,;           . TYPE
                    OM$GENERAL,;        . OWNER ACCESS
                    OM$NONE,;           . OTHER ACCESS
                    OM$EXTENDED,;       . MODE
                    OM$PROGRAM,;        . LOCALITY
                    OM$PUBLIC,;         . ACCESS CONTROL
                    OM$MSU,;            . STORAGE
                    0,;                 . ALIGNMENT
                    0,;                 . ISP LOCAL PRIORITY (UNUSED)
                    OM$UNDEFINED)       . MERGE ORDER (ANY)
$(1)       $BANK OM$CODEG_EMB,01000     .
$(1)       $lit                         . Literals in the code bank
           om$use_lv,17 'LINKBNK',,b2   . define a link vector bank
pardes     $form     2,1,1,23,1,1,1,6   . IP,L,U,S & Res.(0),E,R,W,B
zeros      +         0                  . low value
.
. Expected results
.
ucs2ccsno  $equ      61                 .
ucs2sdfno  $equ      61                 .
ucs2isono  $equ      0                  .
ucs2bitsz  $equ      18                 .
ucs2nmlen  $equ      5                  .
ucs2namew0 +         'UCS-'             .
ucs2namew1 +         '2   '             .
```

```
ucs2namew2 +        '    '          .
ucs2namew3 +        '    '          .
.
jisccsno   $equ     160             .
jissdfno   $equ     5               .
jisisono   $equ     0               .
jisbitsz   $equ     9               .
jisnmlen   $equ     9               .
jisnamew0  +        'JIS-'          .
jisnamew1  +        'X020'          .
jisnamew2  +        '1   '          .
jisnamew3  +        '    '          .
.
de646ccsno $equ     20              .
de646sdfno $equ     20              .
de646isono $equ     210             .
de646bitsz $equ     9               .
de646nmlen $equ     8               .
de646namew0 +       'ISO6'          .
de646namew1 +       '46DE'          .
de646namew2 +       '    '          .
de646namew3 +       '    '          .
.
asciiccsno $equ     1               .
asciisdfno $equ     1               .
asciiisono $equ     60              .
asciibitsz $equ     9               .
asciinmlen $equ     5               .
asciinamew0 +       'ASCI'          .
asciinamew1 +       'I   '          .
asciinamew2 +       '    '          .
asciinamew3 +       '    '          .
.
enuslocno  $equ     42              .
enusloclen $equ     11              .
enuslocnamw0 +      'en_U'          .
enuslocnamw1 +      'S.AS'          .
enuslocnamw2 +      'CII '          .
enuslocnamw3 +      '    '          .
.
psxlocno   $equ     35              .
psxloclen  $equ     5               .
psxlocnamw0 +       'POSI'          .
psxlocnamw1 +       'X   '          .
psxlocnamw2 +       '    '          .
psxlocnamw3 +       '    '          .
.
i8859ccsno $equ     35              .
i8859sdfno $equ     35              .
i8859isono $equ     1000            .
i8859bitsz $equ     9               .
i8859nmlen $equ     9               .
```

```
i8859namew0  +       'ISO8'            .
i8859namew1  +       '859-'            .
i8859namew2  +       '1   '            .
i8859namew3  +       '    '            .
.
nmnbva     +         emsm$nmnb         . VA of EMSM$NMNB
islpktlen  $equ      100               . packet length
islpktcver $equ      1                 . current packet version
islpktcverp1 $equ    islpktcver+1      . current version plus 1
.
.  Extended Mode MASM NMNB Information Packet EQUFs
.
islid      $equf     0,,s4             . NMNB identifier
islcbitsz  $equf     0,,s5             . CCS/Locale character bit size
islversion $equf     0,,s6             . NMNB packet version
islnumber  $equf     01,,w             . CCS/Locale number passed as
input
islccsnml  $equf     02,,w             . CCS Name length
islccsnb   $equf     03,,w             . CCS number
islccsisonb $equf    04,,w             . CCS ISO number
islccssdfnb $equf    05,,w             . CCS SDF number
islccsnmw0 $equf     06,,w             . CCS name (characters 01-04)
islccsnmw1 $equf     07,,w             . CCS name (characters 05-08)
islccsnmw2 $equf     010,,w            . CCS name (characters 09-12)
islccsnmw3 $equf     011,,w            . CCS name (characters 13-16)
isllocnml  $equf     012,,w            . LOCALE name length
isllocnb   $equf     013,,w            . LOCALE number
isllocnmw0 $equf     014,,w            . LOCALE name (characters 01-04)
isllocnmw1 $equf     015,,w            . LOCALE name (characters 05-08)
isllocnmw2 $equf     016,,w            . LOCALE name (characters 09-12)
isllocnmw3 $equf     017,,w            . LOCALE name (characters 13-16)
.
. Skeleton of parameter list passed to emsm$nmnb:
.   Parameter 1: passed by reference , packet VA & length
.
paralst
          pardes     0,0,1,0,0,1,1,0   . parameter descriptor
packetloc +          0                 . VA of NMNB packet
          +          36*islpktlen      . ISL packet length in bits
paralen   $equ       $-paralst         . length of parameter list
alssiz    $equ       islpktlen+paralen . both parameter list and ISL
packet
                                       . will be allocated in ALS
aprint$   $equ       070               . define ER APRINT$
.
. Messages
.
msg11     'Abnormal return from CCS number input
' .
msg12     'NMNB packet contains unexpected results (CCS number input)
' .
msg13     'CCS number input test passed
```

```
' .
.
msg21     'Abnormal return from CCS ISO number input
' .
msg22     'NMNB packet contains unexpected results (CCS ISO number
input)  ' .
msg23     'CCS ISO number input test passed
' .
.
msg31     'Abnormal return from CCS SDF number input
' .
msg32     'NMNB packet contains unexpected results (CCS SDF number
input)  ' .
msg33     'CCS SDF number input test passed
' .
.
msg41     'Abnormal return from Locale number input
' .
msg42     'NMNB packet contains unexpected results (Locale number
input)   ' .
msg43     'Locale number input test passed
' .
.
msg51     'Abnormal return from CCS name input
' .
msg52     'NMNB packet contains unexpected results (CCS name input)
' .
msg53     'CCS name input test passed
' .
.
msg61     'Abnormal return from Locale name input
' .
msg62     'NMNB packet contains unexpected results (Locale name input)
' .
msg63     'Locale name input test passed
' .
.
start*                                .
          buy       *alssiz,*x10,b1   . buy an ALS frame
          lr,u      r1,paralen        . get parameter list length
          la,u      a1,paralst        . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set the source increment
          lxsi,u    a2,1              . set the destination increment
          bt        a2,b1,*a1,b0,*0   . move the parameter list
.
          sbu       b1,x2             . get the BDI of ALS
          la        a2,x10            . get receiver bank offset
          aa,u      a2,paralen        . get packet offset
          lxm       x2,a2             . set VA of packet
          sx        x2,1,x10,b1       .  store into param list
.
```

```
test1                                   .
          lr,u      r1,islpktlen        . get parameter list length
          la,u      a1,zeros            . get sender bank offset
          lxsi,u    a1,0                . set source increment
          la        a2,x10              . get receiver bank offset
          aa,u      a2,paralen          . get packet offset
          lxsi,u    a2,1                . set destination increment
          bt        a2,b1,*a1,b0,*0     . clear the ISLPKT
          lx        x2,x10              . set appropriate
          ax,u      x2,paralen          .    packet values
          la,u      a3,1                . get CCS number identifier
          sa        a3,islid,x2,b1      . set identifier
          la,u      a3,islpktcver       . get current packet version
          sa        a3,islversion,x2,b1 . set version
          la,u      a3,ucs2ccsno        . get CCS number
          sa        a3,islnumber,x2,b1  . set CCS number
          dsl       a2,72               . clear registers A2, A3
          la,u      a0,1                . get EMSM$NMNB number of
parameters
          sbu       b1,a1               .
          aa        a1,x10              . get parameter list address
          lx,u      x2,nmnbva           . load address of EMSM$NMNB
          call      0,x2,b0             . CALL EMSM$NMNB
          lssl      a0,18               . get rid of ELMS sev. and
component info
          jz        a0,check1           . check results if no error
.
nmnberr1                                . non-zero status returned
          sbu       b0,a0               . store code bank VA
          lbu       b13,a0              . base code bank on B13
          lxi,u     a0,0400000          . load basic mode void VA
          lxm,u     a0,$AP(print11)     . load offset to basic mode code
          call      a0                  . call basic mode
          j         test2               . next part
.
check1                                  . (check returned value)
          la        a2,x10              . get offset
          aa,u      a2,paralen          . get to packet offset
          la        a3,islcbitsz,a2,b1  . get character bit size
          te,u      a3,ucs2bitsz        . bit size correct?
          j         checkerr1           . no, issue error
          la        a3,islccsnml,a2,b1  . yes, get CCS name length
          te,u      a3,ucs2nmlen        . CCS name length correct?
          j         checkerr1           . no, issue error
          la        a3,islccsnb,a2,b1   . yes, get CCS number
          te,u      a3,ucs2ccsno        . CCS number correct?
          j         checkerr1           . no, issue error
          la        a3,islccsisonb,a2,b1  . yes, get CCS ISO number
          te,u      a3,ucs2isono        . CCS ISO number correct?
          j         checkerr1           . no, issue error
          la        a3,islccssdfnb,a2,b1 . yes, get CCS SDF number
          te,u      a3,ucs2sdfno        . CCS SDF number correct?
```

```
          j         checkerr1          . no, issue error
          la        a3,islccsnmw0,a2,b1 . yes, get CCS name char 01-04
          te        a3,ucs2namew0,,b0 . characters 01-04 correct?
          j         checkerr1          . no, issue error
          la        a3,islccsnmw1,a2,b1 . yes, get CCS name char 05-08
          te        a3,ucs2namew1,,b0 . characters 05-08 correct?
          j         checkerr1          . no, issue error
          la        a3,islccsnmw2,a2,b1   . yes, get CCS name char 09-
12
          te        a3,ucs2namew2,,b0 . characters 09-12 correct?
          j         checkerr1          . no, issue error
          la        a3,islccsnmw3,a2,b1   . yes, get CCS name char 13-
16
          te        a3,ucs2namew3,,b0 . characters 13-16 correct?
          j         checkerr1          . no, issue error
          j         checkok1           . yes, test passed
checkerr1                              .
          sbu       b0,a0              . store code bank VA
          lbu       b13,a0             . base code bank on B13
          lxi,u     a0,0400000         . load basic mode void VA
          lxm,u     a0,$AP(print12)    . load offset to basic mode code
          call      a0                 . call basic mode
          j         test2              . check CCS ISO number input
checkok1                               .
          sbu       b0,a0              . store code bank VA
          lbu       b13,a0             . base code bank on B13
          lxi,u     a0,0400000         . load basic mode void VA
          lxm,u     a0,$AP(print13)    . load offset to basic mode code
          call      a0                 . call basic mode
.
test2     lr,u      r1,islpktlen       . get parameter list length
          la,u      a1,zeros           . get sender bank offset
          lxsi,u    a1,0               . set the increments
          la        a2,x10             . get receiver bank offset
          aa,u      a2,paralen         . get packet offset
          lxsi,u    a2,1               . get source increment
          bt        a2,b1,*a1,b0,*0    . clear the packet
          la        a2,x10             . set appropriate
          aa,u      a2,paralen         .   packet values
          la,u      a3,2               . get CCS ISO identifier
          sa        a3,islid,a2,b1     . save identifier
          la,u      a3,islpktcver      . get current version
          sa        a3,islversion,a2,b1 . save current version
          la,u      a3,210             . get CCS ISO number
          sa        a3,islnumber,a2,b1 . set CCS ISO number input
          dsl       a2,72              . clear registers A2, A3
          la,u      a0,1               . get number of parameters
          sbu       b1,a1              . store base register
          aa        a1,x10             . update to parameter list
          lx,u      x2,nmnbva          . get VA of EMSM$NMNB
          call      0,x2,b0            . CALL EMSM$NMNB
          lssl      a0,18              . get rid of ELMS severity & comp
```

```
ID info
          jz        a0,check2          . jump if error was not returned
nmnberr2                               . non-zero status returned
          sbu       b0,a0              . store code bank VA
          lbu       b13,a0             . base code bank on B13
          lxi,u     a0,0400000         . load basic mode void VA
          lxm,u     a0,$AP(print21)    . load offset to basic mode code
          call      a0                 . call basic mode
          j         test3              . test CCS SDF number input
.
check2                                 . (check returned values)
          la        a2,x10             . get ALS address
          aa,u      a2,paralen         . get to packet offset
          la        a3,islcbitsz,a2,b1 . get char-bit-size
          te,u      a3,de646bitsz      . is bit size correct?
          j         checkerr2          . no, issue error
          la        a3,islccsnml,a2,b1 . yes, get CCS name length
          te,u      a3,de646nmlen      . is CCS name length correct?
          j         checkerr2          . no, issue error
          la        a3,islccsnb,a2,b1  . yes, get CCS number
          te,u      a3,de646ccsno      . is CCS number correct?
          j         checkerr2          . no, issue error
          la        a3,islccsisonb,a2,b1 . yes, get CCS ISO number
          te,u      a3,de646isono      . is CCS ISO number correct?
          j         checkerr2          . no, issue error
          la        a3,islccssdfnb,a2,b1 . yes, get CCS SDF number
          te,u      a3,de646sdfno      . is CCS SDF number correct?
          j         checkerr2          . no, issue error
          la        a3,islccsnmw0,a2,b1 . yes, get CCS name characters
01-04
          te        a3,de646namew0,,b0 . CCS name characters 01-04
correct?
          j         checkerr2          . no, issue error
          la        a3,islccsnmw1,a2,b1 . yes, get CCS name characters
05-08
          te        a3,de646namew1,,b0 .  CCS name characters 05-08
correct?
          j         checkerr2          . no, issue error
          la        a3,islccsnmw2,a2,b1 . yes, get CCS name characters
09-12
          te        a3,de646namew2,,b0 . CCS name characters 09-12
correct?
          j         checkerr2          . no, issue error
          la        a3,islccsnmw3,a2,b1 . yes, get CCS name characters
13-16
          te        a3,de646namew3,,b0 . CCS name characters 13-16
correct?
          j         checkerr2          . no, issue error
          j         checkok2           . yes, results are OK
checkerr2                              .
          sbu       b0,a0              . store code bank va
          lbu       b13,a0             . base code bank on b13
```

```
          lxi,u     a0,0400000        . load basic mode void va
          lxm,u     a0,$AP(print22)   . load offset to basic mode code
          call      a0                . call basic mode
          j         test3             . next part
checkok2                              .
          sbu       b0,a0             . store code bank va
          lbu       b13,a0            . base code bank on b13
          lxi,u     a0,0400000        . load basic mode void va
          lxm,u     a0,$AP(print23)   . load offset to basic mode code
          call      a0                . call basic mode
.
test3     lr,u      r1,islpktlen      . get parameter list length
          la,u      a1,zeros          . get sender bank offset
          lxsi,u    a1,0              . set the increments
          la        a2,x10            . get receiver bank offset
          aa,u      a2,paralen        . get packet offset
          lxsi,u    a2,1              . get source increment
          bt        a2,b1,*a1,b0,*0   . clear the packet
          la        a2,x10            . set appropriate
          aa,u      a2,paralen        .   packet values
          la,u      a3,3              . get CCS SDF number identifier
          sa        a3,islid,a2,b1    . save identifier
          la,u      a3,islpktcver     . get current packet version
          sa        a3,islversion,a2,b1 . save current packet version
          la,u      a3,61             . get CCS SDF number
          sa        a3,islnumber,a2,b1 . save number in packet
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . get number of parameters
          sbu       b1,a1             . save base register
          aa        a1,x10            . get address of param list
          lx,u      x2,nmnbva         . get VA of EMSM$NMNB
          call      0,x2,b0           . call EMSM$NMNB
          lssl      a0,18             . shift off ELMS severity & comp
ID info
          jz        a0,check3         . jump if no error returned
nmnberr3                              . non-zero status returned
          sbu       b0,a0             . store code bank VA
          lbu       b13,a0            . base code bank on B13
          lxi,u     a0,0400000        . load basic mode void VA
          lxm,u     a0,$AP(print31)   . load offset to basic mode code
          call      a0                . call basic mode
          j         test4             . next part
.
check3                                . (check returned value)
          la        a2,x10            . get ALS address
          aa,u      a2,paralen        . get packet address
          la        a3,islcbitsz,a2,b1 . get character bit size
          te,u      a3,ucs2bitsz      . bit size correct?
          j         checkerr3         . no, issue error
          la        a3,islccsnml,a2,b1 . yes, get CCS name length
          te,u      a3,ucs2nmlen      . CCS name length correct?
          j         checkerr3         . no, issue error
```

```
          la        a3,islccsnb,a2,b1 . yes, get CCS number
          te,u      a3,ucs2ccsno      . CCS number correct?
          j         checkerr3         . no, issue error
          la        a3,islccsisonb,a2,b1  . yes, get CCS ISO number
          te,u      a3,ucs2isono      . CCS ISO number correct?
          j         checkerr3         . no, issue error
          la        a3,islccssdfnb,a2,b1 . get CCS SDF number
          te,u      a3,ucs2sdfno      . CCS SDF number correct?
          j         checkerr3         . no, issue error
          la        a3,islccsnmw0,a2,b1 . yes, get CCS name characters
01-04
          te        a3,ucs2namew0,,b0 .  characters 01-04 correct?
          j         checkerr3         . no, issue error
          la        a3,islccsnmw1,a2,b1 . yes, get CCS name characters
05-08
          te        a3,ucs2namew1,,b0 .  characters 05-08 correct?
          j         checkerr3         . no, issue error
          la        a3,islccsnmw2,a2,b1 . yes, get CCS name characters
09-12
          te        a3,ucs2namew2,,b0 . characters 09-12 correct?
          j         checkerr3         . no, issue error
          la        a3,islccsnmw3,a2,b1 . yes, get CCS name characters
13-16
          te        a3,ucs2namew3,,b0 . characters 13-16 correct?
          j         checkerr3         . no, issue error
          j         checkok3          . yes
checkerr3                             .
          sbu       b0,a0             . store code bank va
          lbu       b13,a0            . base code bank on b13
          lxi,u     a0,0400000        . load basic mode void va
          lxm,u     a0,$AP(print32)   . load offset to basic mode code
          call      a0                . call basic mode
          j         test4             . next part
checkok3                              .
          sbu       b0,a0             . store code bank va
          lbu       b13,a0            . base code bank on b13
          lxi,u     a0,0400000        . load basic mode void va
          lxm,u     a0,$AP(print33)   . load offset to basic mode code
          call      a0                . call basic mode
.
test4     lr,u      r1,islpktlen      . get parameter list length
          la,u      a1,zeros          . get sender bank offset
          lxsi,u    a1,0              . set the increments
          la        a2,x10            . get receiver bank offset
          aa,u      a2,paralen        . get packet offset
          lxsi,u    a2,1              . get increment
          bt        a2,b1,*a1,b0,*0   . clear the packet
          la        a2,x10            . set appropriate
          aa,u      a2,paralen        .   packet values
          la,u      a3,4              . get Locale name identifier
          sa        a3,islid,a2,b1    . set Locale name identifier
          la,u      a3,islpktcver     . get current version
```

```
            sa      a3,islversion,a2,b1 . save current version
            la,u    a3,42               . get locale number
            sa      a3,islnumber,a2,b1 . set locale number input
            dsl     a2,72               . clear registers A2, A3
            la,u    a0,1                . number of parameters
            sbu     b1,a1               . save base register
            aa      a1,x10              . where the param list is
            lx,u    x2,nmnbva           . get address of EMSM$NMNB
            call    0,x2,b0             . call EMSM$NMNB
            lssl    a0,18               . shift off ELMS information
            jz      a0,check4           . jump if no error
nmnberr4                                . non-zero status returned
            sbu     b0,a0               . store code bank va
            lbu     b13,a0              . base code bank on b13
            lxi,u   a0,0400000          . load basic mode void va
            lxm,u   a0,$AP(print41)     . load offset to basic mode code
            call    a0                  . call basic mode
            j       test5               . next part
.
check4                                  . (check returned value)
            la      a2,x10              . get ALS address
            aa,u    a2,paralen          . get packet address
            la      a3,islcbitsz,a2,b1 . get char-bit-size
            te,u    a3,asciibitsz       . is bit size correct?
            j       checkerr4           . no, issue error
            la      a3,islccsnml,a2,b1 . yes, get CCS name length
            te,u    a3,asciinmlen       . is name length correct?
            j       checkerr4           . no, issue error
            la      a3,islccsnb,a2,b1 . yes, get CCS number
            te,u    a3,asciiccsno       . is CCS number correct?
            j       checkerr4           . no, issue error
            la      a3,islccsisonb,a2,b1 . yes, get CCS ISO number
            te,u    a3,asciiisono       . is CCS ISO number correct?
            j       checkerr4           . no, issue error
            la      a3,islccssdfnb,a2,b1 . yes, get CCS SDF number
            te,u    a3,asciisdfno       . is CCS SDF number correct?
            j       checkerr4           . no, issue error
            la      a3,islccsnmw0,a2,b1 . yes, get CCS name characters
01-04
            te      a3,asciinamew0,,b0 . characters 01-04 correct?
            j       checkerr4           . no, issue error
            la      a3,islccsnmw1,a2,b1 . yes, get CCS name characters
05-08
            te      a3,asciinamew1,,b0 . CCS name characters 05-08
correct?
            j       checkerr4           . no, issue error
            la      a3,islccsnmw2,a2,b1 . yes, get CCS name characters
09-12
            te      a3,asciinamew2,,b0 . CCS name characters 09-12
correct?
            j       checkerr4           . no, issue error
            la      a3,islccsnmw3,a2,b1 . yes, get CCS name characters
```

```
13-16
          te        a3,asciinamew3,,b0 . CCS name characters 13-16
correct?
          j         checkerr4          . no, issue error
          la        a3,isllocnml,a2,b1 . yes, get locale name length
          te,u      a3,enusloclen      . locale name length correct?
          j         checkerr4          . no, issue error
          la        a3,isllocnb,a2,b1  . yes, get locale number
          te,u      a3,enuslocno       . locale number correct?
          j         checkerr4          . no, issue error
          la        a3,isllocnmw0,a2,b1 . yes, get locale name
characters 01-04
          te        a3,enuslocnamw0,,b0 . locale name characters 01-04
correct?
          j         checkerr4          . no, issue error
          la        a3,isllocnmw1,a2,b1 . yes, get locale name
characters 05-08
          te        a3,enuslocnamw1,,b0 . locale name characters 05-08
correct?
          j         checkerr4          . no, issue error
          la        a3,isllocnmw2,a2,b1 . yes, get locale name
characters 09-12
          te        a3,enuslocnamw2,,b0 . locale name characters 09-12
correct?
          j         checkerr4          . no, issue error
          la        a3,isllocnmw3,a2,b1 . yes, get locale name
characters 13-16
          te        a3,enuslocnamw3,,b0 . locale name characters 13-16
correct?
          j         checkerr4          . no, issue error
          j         checkok4           . yes,
checkerr4                              .
          sbu       b0,a0              . store code bank VA
          lbu       b13,a0             . base code bank on B13
          lxi,u     a0,0400000         . load basic mode void VA
          lxm,u     a0,$AP(print42)    . load offset to basic mode code
          call      a0                 . call basic mode
          j         test5              . next test
checkok4                               .
          sbu       b0,a0              . store code bank VA
          lbu       b13,a0             . base code bank on B13
          lxi,u     a0,0400000         . load basic mode void VA
          lxm,u     a0,$AP(print43)    . load offset to basic mode code
          call      a0                 . call basic mode
.
test5     lr,u      r1,islpktlen       . get parameter list length
          la,u      a1,zeros           . get sender bank offset
          lxsi,u    a1,0               . set the increments
          la        a2,x10             . get receiver bank offset
          aa,u      a2,paralen         . get packet offset
          lxsi,u    a2,1               . get increment
          bt        a2,b1,*a1,b0,*0    . clear the packet
```

```
          la        a2,x10            . set appropriate
          aa,u      a2,paralen        .  packet values
          la,u      a3,5              . get CCS name identifier
          sa        a3,islid,a2,b1    . save identifier
          la,u      a3,islpktcver     . get current version
          sa        a3,islversion,a2,b1 . save current version
          la,u      a3,9              . get CCS name length
          sa        a3,islccsnml,a2,b1 . set CCS name length input
          la        a3,jisnamew0,,b0  . get CCS name characters 01-04
          sa        a3,islccsnmw0,a2,b1 . set CCS name characters 01-04
          la        a3,jisnamew1,,b0  . get CCS name characters 05-08
          sa        a3,islccsnmw1,a2,b1 . set CCS name characters 05-08
          la        a3,jisnamew2,,b0  . get CCS name characters 09-12
          sa        a3,islccsnmw2,a2,b1 . set CCS name characters 09-12
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          sbu       b1,a1             . save base register
          aa        a1,x10            . where the param list is
          lx,u      x2,nmnbva         . load address of EMSM$NMNB
          call      0,x2,b0           . call EMSM$NMNB
          lssl      a0,18             . shift off ELMS severity comp ID
info
          jz        a0,check5         . jump if no error
nmnberr5                              . non-zero status returned
          sbu       b0,a0             . store code bank VA
          lbu       b13,a0            . base code bank on B13
          lxi,u     a0,0400000        . load basic mode void VA
          lxm,u     a0,$AP(print51)   . load offset to basic mode code
          call      a0                . call basic mode
          j         test5             . next test
.
check5                                . (check returned value)
          la        a2,x10            . get ALS address
          aa,u      a2,paralen        . get packet address
          la        a3,islcbitsz,a2,b1 . get char bit size
          te,u      a3,jisbitsz       . character bit size correct?
          j         checkerr5         . no, issue error
          la        a3,islccsnb,a2,b1 . get CCS number
          te,u      a3,jisccsno       . CCS number correct
          j         checkerr5         . no, issue error
          la        a3,islccsisonb,a2,b1 . yes, get CCS ISO number
          te,u      a3,jisisono       . CCS ISO number correct?
          j         checkerr5         . no, issue error
          la        a3,islccssdfnb,a2,b1 . yes, get CCS SDF number
          te,u      a3,jissdfno       . CCS SDF number correct?
          j         checkerr5         . no, issue error
          la        a3,islccsnmw0,a2,b1 . yes, get CCS name characters
01-04
          te        a3,jisnamew0,,b0  . CCS name characters 01-04
correct?
          j         checkerr5         . no, issue error
          la        a3,islccsnmw1,a2,b1 . yes, get CCS name characters
```

```
05-08
          te        a3,jisnamew1,,b0  . CCS name characters 05-08
correct?
          j         checkerr5         . no, issue error
          la        a3,islccsnmw2,a2,b1 . yes, get CCS name characters
09-12
          te        a3,jisnamew2,,b0  . CCS name characters 09-12
correct?
          j         checkerr5         . no, issue error
          la        a3,islccsnmw3,a2,b1 . yes, get CCS name characters
13-16
          te        a3,jisnamew3,,b0  . CCS name characters 13-16
correct?
          j         checkerr5         . no, issue error
          j         checkok5          . yes,
checkerr5                             .
          sbu       b0,a0             . store code bank VA
          lbu       b13,a0            . base code bank on B13
          lxi,u     a0,0400000        . load basic mode void VA
          lxm,u     a0,$AP(print52)   . load offset to basic mode code
          call      a0                . call basic mode
          j         test6             . next test
checkok5                              .
          sbu       b0,a0             . store code bank VA
          lbu       b13,a0            . base code bank on B13
          lxi,u     a0,0400000        . load basic mode void VA
          lxm,u     a0,$AP(print53)   . load offset to basic mode code
          call      a0                . call basic mode
.
test6     lr,u      r1,islpktlen      . get parameter list length
          la,u      a1,zeros          . get sender bank offset
          lxsi,u    a1,0              . set the increments
          la        a2,x10            . get receiver bank offset
          aa,u      a2,paralen        . get packet offset
          lxsi,u    a2,1              . set increment
          bt        a2,b1,*a1,b0,*0   . clear the packet
          la        a2,x10            . set appropriate
          aa,u      a2,paralen        .  packet values
          la,u      a3,6              . get locale name identifier
          sa        a3,islid,a2,b1    . set identifier
          la,u      a3,islpktcver     . get current version
          sa        a3,islversion,a2,b1 . save current version
          la,u      a3,psxloclen      . get locale name length
          sa        a3,isllocnml,a2,b1 . save locale name length input
          la        a3,psxlocnamw0,,b0 . get locale name characters 01-
04
          sa        a3,isllocnmw0,a2,b1 . set locale name character 01-
04
          la        a3,psxlocnamw1,,b0 . get locale name characters 05-
08
          sa        a3,isllocnmw1,a2,b1 . set locale name characters
05-08
```

```
          la        a3,psxlocnamw2,,b0 . get locale name characters 09-
12
          sa        a3,isllocnmw2,a2,b1 . set locale name characters
09-12
          la        a3,psxlocnamw3,,b0 . get locale name chracters 13-
16
          sa        a3,isllocnmw3,a2,b1 . set locale name characters
13-16
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,1              . number of parameters
          sbu       b1,a1             . save base register
          aa        a1,x10            . where the param list is
          lx,u      x2,nmnbva         . load address of EMSM$NMNB
          call      0,x2,b0           . call EMSM$NMNB
          lssl      a0,18             . shift off ELMS severity, comp
ID info
          jz        a0,check6         . jump if no error
nmnberr6                              . non-zero status returned
          sbu       b0,a0             . store code bank VA
          lbu       b13,a0            . base code bank on B13
          lxi,u     a0,0400000        . load basic mode void VA
          lxm,u     a0,$AP(print61)   . load offset to basic mode code
          call      a0                . call basic mode
          j         testend           . next part
.
check6                                . (check returned value)
          la        a2,x10            . get receiver bank offset
          aa,u      a2,paralen        . get packet offset
          la        a3,islcbitsz,a2,b1 . get character bit size
          te,u      a3,i8859bitsz     . character bit size correct?
          j         checkerr6         . no, issue error
          la        a3,islccsnml,a2,b1 . yes, get CCS name length
          te,u      a3,i8859nmlen     . CCS name length correct?
          j         checkerr6         . no, issue error
          la        a3,islccsnb,a2,b1 . yes, get CCS number
          te,u      a3,i8859ccsno     . CCS number correct?
          j         checkerr6         . no, issue error
          la        a3,islccsisonb,a2,b1 . get CCS ISO number
          te,u      a3,i8859isono     . CCS ISO number correct?
          j         checkerr6         . no, issue error
          la        a3,islccssdfnb,a2,b1 . yes, get CCS SDF number
          te,u      a3,i8859sdfno     . CCS SDF number correct?
          j         checkerr6         . no, issue error
          la        a3,islccsnmw0,a2,b1 . get CCS name characters 01-04
          te        a3,i8859namew0,,b0 . CCS name characters 01-04
correct?
          j         checkerr6         . no, issue error
          la        a3,islccsnmw1,a2,b1 . yes, get CCS name characters
05-08
          te        a3,i8859namew1,,b0 . CCS name characters 05-08
correct?
          j         checkerr6         . no, issue error
```

```
          la        a3,islccsnmw2,a2,b1 . yes, get CCS name characters
09-12
          te        a3,i8859namew2,,b0 . CCS name characters 09-12
correct?
          j         checkerr6          . no, issue error
          la        a3,islccsnmw3,a2,b1 . yes, get CCS name characters
13-16
          te        a3,i8859namew3,,b0 . CCS name characters 13-16
correct?
          j         checkerr6          . no, issue error
          la        a3,isllocnb,a2,b1  . get locale number
          te,u      a3,psxlocno        . locale number correct?
          j         checkerr6          . no, issue error
          j         checkok6           . yes,
checkerr6                              .
          sbu       b0,a0              . store code bank VA
          lbu       b13,a0             . base code bank on B13
          lxi,u     a0,0400000         . load basic mode void VA
          lxm,u     a0,$AP(print62)    . load offset to basic mode code
          call      a0                 . call basic mode
          j         testend            .
checkok6                               .
          sbu       b0,a0              . store code bank VA
          lbu       b13,a0             . base code bank on B13
          lxi,u     a0,0400000         . load basic mode void VA
          lxm,u     a0,$AP(print63)    . load offset to basic mode code
          call      a0                 . call basic mode
.
testend   ax,u      x10,alssiz         . sell ALS frame
          rtn                          . return to caller


.
          $basic                       . generate basic mode
print11                                .
          lxi,u     a0,0120            . get length of message
          lxm,u     a0,msg11           . get message address
          er        aprint$            . print the msg in ASCII
          lbj       x11,x11            .
print12                                .
          lxi,u     a0,0120            . get length of message
          lxm,u     a0,msg12           . get message address
          er        aprint$            . print the msg in ASCII
          lbj       x11,x11            .
print13                                .
          lxi,u     a0,0120            . get length of message
          lxm,u     a0,msg13           . get message address
          er        aprint$            . print the msg in ASCII
          lbj       x11,x11            .
.
print21                                .
          lxi,u     a0,0120            . get length of message
          lxm,u     a0,msg21           . get message address
```

```
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
print22                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg22         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
print23                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg23         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
.
print31                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg31         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
print32                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg32         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
print33                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg33         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
.
print41                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg41         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
print42                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg42         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
print43                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg43         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
.
print51                              .
          lxi,u     a0,0120          . get length of message
          lxm,u     a0,msg51         . get message address
          er        aprint$          . print the msg in ASCII
          lbj       x11,x11          .
print52                              .
          lxi,u     a0,0120          . get length of message
```

```
            lxm,u     a0,msg52          . get message address
            er        aprint$           . print the msg in ASCII
            lbj       x11,x11           .
print53                                 .
            lxi,u     a0,0120           . get length of message
            lxm,u     a0,msg53          . get message address
            er        aprint$           . print the msg in ASCII
            lbj       x11,x11           .
.
print61                                 .
            lxi,u     a0,0120           . get length of message
            lxm,u     a0,msg61          . get message address
            er        aprint$           . print the msg in ASCII
            lbj       x11,x11           .
print62                                 .
            lxi,u     a0,0120           . get length of message
            lxm,u     a0,msg62          . get message address
            er        aprint$           . print the msg in ASCII
            lbj       x11,x11           .
print63                                 .
            lxi,u     a0,0120           . get length of message
            lxm,u     a0,msg63          . get message address
            er        aprint$           . print the msg in ASCII
            lbj       x11,x11           .
.
            $end      start             . end of test program
```

6.29. NL_LANGINFO Service Routine

Entry Point

EMSM$NLLANG

Description

NL_LANGINFO returns a string containing relevant information about the particular language or cultural area defined in the program's locale.

NL_LANGINFO requires a specific locale item number so it can determine what locale information to return.

Calling Sequence

```
CALL      X11,EMSM$NLLANG
```

Parameter List

In the standard calling sequence, each parameter format is a two-word entry. The following is required NL_LANGINFO input for these entries:

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the virtual address to the parameter list formatted in the standard calling sequence. The following describes those parameters:

Parameter 1

indicates which locale item to retrieve. Variables equated to possible values for this field are automatically defined when ISL-PKT-DEFS/MSM is included.

Parameter Name: item
Parameter Type: Immediate Data

Parameter 2

is the output buffer where the resulting string is written.

Parameter Name: *str_out*
Parameter Type: Ordinary Data
Required Access: Read, Write

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the bit length of the resulting string if A0 indicates normal completion.

In a locale where the particular locale is not defined, NL_LANGINFO returns a string to the correponding string in the POSIX locale.

The character length of the resulting string is also returned in Register A1.

In all locales, NL_LANGINFO returns an error if item contains an invalid status value.

Error Statuses

NL_LANGINFO can return the following error statuses in the *error* field:

- 0 – ISOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR

- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREADx
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT
- 50 – ISLINVLOCITM

Sample Code

NL_LANGINFO is used to return a string containing information about a particular language or cultural area defined in the program's locale. This example illustrates a call to return the full name of the first day in the week, based on the POSIX locale, which specifies the minimal environment for C-language translation.

```
        $include  'maxr$/'          .
           $include  'om$def'           . OM definitions
           $include  'cbep$$isl-em'     . I18NLIB entry point defs
           $include  'isl-pkt-defs/msm' .
           $extend                      . indicate extended mode
           $obj                         .
           $ascii                       .
pardes     $form     2,1,1,23,1,1,1,6   . IP,L,U,S & RES.(0),E,R,W,B
.
$(0),dbnk $bank om$data_emb 'emdata'   .
           $lit                         . literals to loc counter 0
outbufl    $equ      5                  . # of output buffer words
outbuf     $res      outbufl            . space for resulting str
.
pars       pardes    2,0,0,0,0,0,0,0    . parameter 1 descriptor
           +         isl_day_1          . item number
           +         0                  . ignored
           pardes    0,0,0,0,0,1,1,0    . parameter 2 descriptor
           +         0                  . VA to start of arguments
           +         outbufl*36         . bit length of buffer
parslen    $equ      $-pars             . length of parameter list
.
$(1)       $bank om$code_emb 'emstart' .
           om$use_lv,17 'lv',,B2        . define link vector bank
start                                   .
           lbu       b2,r0              . base link vector bank on b2
           om$load_bdr b3,dbnk          . base data on b3
           buy       *parslen,*x10,b1   . buy an ALS frame
           la,u      a1,pars            . source addr (param list)
```

```
            lxsi,u    a1,1              . incrementor
            la        a2,x10            . destination on ALS
            lxsi,u    a2,1              . incrementor
            lr,u      r1,parslen        . # words to xfer to ALS
            bt        a2,b1,*a1,b3,*0   . move param list to ALS
            sbu       b3,a1             . DBANK L,BDI,offset
            aa,u      a1,outbuf         . VA of output buffer in A1
            sa        a2,*4,x10,b1      . save into param on ALS
            sbu       b1,a1             . ALS L,BDI,offset
            aa        a1,x10            . VA to parameter list
            la,u      a0,2              . # of parameters to pass
            dsl       a2,72             . clear registers A2, A3
            om$call   'EMSM$NLLANG'     . call NL_LANGINFO
            te        a0,(045625000000),,B9 . error returned?
            j         processerr        . yes
.
. Normal processing
.
            j         comrtn            . jump to common return
processerr                              .
.
. Error processing
.
comrtn                                  .
            ax,u      x10,parslen       . sell ALS frame
                .
                .
                .
```

The following output for this locale is returned as follows:

```
Sunday
```

6.30. OPEN_CCS_TO_CCS Service Routine

Entry Points

EMSM$OPENCCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

Calling Sequence

```
openccsva +       EMSM$OPENCCS      . VA of OPEN_CCS_TO_CCS
            LA,U    A0,2              . two parameters in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,openccsva      . load address of service routine
            CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following are the OPEN_CCS_TO_CCS parameters and the required input for each:

from-ccs

is an integer that contains the Unisys CCS number of the *from-string*.

Parameter Type: Immediate Data
Required Access: Read

to-ccs

is an integer that contains the Unisys CCS number of the *to-string*.

Parameter Type: Immediate Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the conversion descriptor (internal identifier for the transliteration tables). This value is passed to CCS_TO_CCS.

Error Statuses

See 6.8 for the error statuses.

Sample Code

See 6.30 for sample code.

6.31. PUTENV Service Routine

Entry Point

EMSM$PUTENV

Description

PUTENV sets the current value of the environment variable for the run level.

Trailing Spaces in Environment Variable Names and Values

Trailing spaces are significant in the names and values of environment variables. For example, a name such as 'LC_COLLATE ' is not the same as 'LC_COLLATE'. PUTENV would match 'LC_COLLATE ' with the I18N environment variable LC_COLLATE.

They can be a part of the environment variable value that PUTENV places in the run control information. Any trailing spaces in this value are placed in the *name=value* string.

Calling Sequence

```
putenvva +        EMSM$PUTENV       . virtual address of PUTENV
            LA,U     A0,2              . two parameters in list
            LA       A1,virtual-address-of-parameter-list
            LX,U     X2,putenvva       . load address of service routine
            CALL     0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, each parameter format is a three-word entry. The following are the PUTENV parameters and the required input for these entries:

environment-variable-name

is the name of the environment variable for which the value is to be set.

Parameter Type: Ordinary Data
Required Access: Read

environment-variable-value

is the value for the environment variable.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Error Statuses

PUTENV can return the following error statuses in Register A0:

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHA
- 21 – ISLVALINVCHA

- 22 – ISLAREAFULL
- 23 – ISLNAMILLTIP
- 25 – ISLVALTOOLON
- 38 – ISLPUTENVERR

Sample Code

PUTENV is used to set environment variables (EV) at the run level. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
putenvva   +         EMSM$PUTENV       . VA of EMSM$PUTENV
evnam      'LC_CTYPE'
evval      'en_US.8859-1'
.
. Skeleton of parameter list passed to EMSM$PUTENV:
.   Parameter 1:  EV name, ordinary data (OD), read access
.
parlskel                              .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of EV name
           +         9*8               . length of name in bits
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of EV value
           +         9*12              . length of value in bits
.
. Define length parameters.
.
parlstlen $equ      $-parlskel          . length of parameter list
als_size  $equ      $parlstlen          . ALS frame size
.
. Code begins here.
.
callputenv*                           .
           buy       *als_size,*x10,b1 . buy ALS frame
           lr,u      r1,parlstlen      . param. list len. to move
           la,u      a1,parlskel       . get sender bank offset
           la        a2,x10            . get receiver bank offset
           lxsi,u    a1,1              . set the increments
           lxsi,u    a2,1              .
           bt        a2,b1,*a1,b0,*0   . move parameter list
           sbu       b0,a2             . get L,BDI of code bank
           aa,u      a2,evnam          . create VA of EV name
           sa        a2,*1,x10,b1      . set EV_NAM VA in par. list
```

```
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,evval          . create VA of EV value
          sa        a2,*4,x10,b1      . set EV_NAM VA in par. list
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of par. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,2              . number of parameters
          lx,u      x2,putenvva       . load VA of putenv
          call      0,x2,b0           . call putenv
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. PUTENV was successful.
.
subexit                             .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. PUTENV returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                          . error processing
          tn        a0                . error or warning?
          j         processit         . warning, process it
                                      . error-putenv failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the non-severe error.
.
processit                           .
.
. Non-severe processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end putenv
```

6.32. SETLOCALE Service Routine

Entry Point

EMSM$SETLOC

Description

SETLOCALE Is used to set category values and query locale information for a user program.

The setting of category values is indeterminate when a user program begins execution. This program calls SETLOCALE to set all or a portion of these values, as specified by the *category* and *requested-locale* fields.

SETLOCALE can query information about all or a portion of the program's locale.

If *requested locale* is set to "_QUERY", SETLOCALE writes the string associated with the category for the current locale into the data area indicated by *resulting locale*. The category's locale is not changed.

In a subsequent call to SETLOCALE, you can include the *resulting locale* string value as the *requested locale* field to restore a category value.

A query of the category LC_ALL returns the locale name for the LC_COLLATE category.

Once called by the program, SETLOCALE establishes the locale for the current activity. This operation includes such functions as specifying (1) the locale name and (2) the length and storage location for this name.

Calling Sequence

```
setlocva +       EMSM$SETLOC        . virtual addr. of SETLOCALE
            LA,U    A0,3               . three parameters in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,setlocva        . load address of service routine
            CALL    0,X2,B0            . call service routine
```

Parameter List

In the standard calling sequence, each parameter format is a three-word entry. The following are the SETLOCALE parameters and the required input for these entries:

Parameter 1

category

is an integer that specifies the portion of the program's locale to be set or queried.

There are seven allowable categories: 0=LC_ALL, 1=LC_COLLATE, 2=LC_CTYPE, 3=LC_MESSAGES, 4=LC_MONETARY, 5=LC_NUMERIC, and 6=LC_TIME.

Parameter Type: Immediate Data
Required Access: Read

Parameter 2

requested-locale

is a string that specifies the name of a locale. This value is either "_DEFAULT" or "_QUERY".

The maximum length for a locale name is 16 characters. When dealing with locale names, SETLOCALE is case-sensitive and ignores trailing spaces.

Parameter Type: Ordinary Data
Required Access: Read

Parameter 3

resulting-locale

is a character variable set by SETLOCALE. This is the name of the locale that is set or queried for in *requested-locale*. If this operation is not successful, the variable is undefined. The data area for *resulting-locale* may be the same as the one passed as *requested-locale*.

Parameter Type: Ordinary Data
Required Access: Read and Write

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the bit length of the value indicated by *resulting-locale*.

Parameter 3

resulting-locale

If the SETLOCALE operation is successful, the area indicated by *resulting-locale* contains the locale name that was set or queried for in *requested-locale*.

If the SETLOCALE operation is not successful, (1) an error status is returned that indicates why the operation failed; (2) the locale of the program is not changed; and (3) the value of the *resulting-locale* string is undefined.

Error Statuses

SETLOCALE can return the following error statuses in Register A0:

- 0-ISLOK
- 3-ISLBUFSHORT
- 15-ISLTOOMANPAR
- 16-ISLTOOFEWPAR
- 20-ISLNAMINVCHR
- 21-ISLVALINVCHR
- 26-ISLLOCNOTFOU
- 28-ISLINVCAT

- 30-ISLCORRUPTAB
- 39-ISLGETENVERR
- 41-ISLASTRREAD
- 42-ISLASTRWRITE
- 43-ISLCREATBANK
- 44-ISLCHPTRESRT

Sample Code

The code example shows how to use SETLOCALE to establish a locale for an activity. For this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
lc_all_cat $equ     0                  . LC_ALL category
catvallen $equ      20                 . word len of return buffer
pardes    $form     2,1,1,23,1,1,1,6   . IP,L,U,S & Res.(0),E,R,W,B
setlocva  +         EMSM$SETLOC        . VA of setlocale
query_req '_QUERY'                     . request value of a category
request_len $equ    $sl('_QUERY')      . char. length of request
.
. Skeleton of parameter list passed to EMSM$SETLOC:
.   Parameter 1:  category, immediate data (ID)
.   Parameter 2:  category_request, ordinary data (OD), read
.   Parameter 3:  category_value, OD, read, write access
.
parlskel                               .
           pardes    2,0,0,0,0,0,0,0   . parameter 1 descriptor
           +         lc_all_cat        . value of LC_ALL category
           +         0                 . unused for 1 word value
           pardes    0,0,0,0,0,1,0,0   . parameter 2 descriptor
           +         0                 . VA of request
           +         request_len*9     . length of request string
           pardes    0,0,0,0,0,1,1,0   . parameter 3 descriptor
           +         0                 . VA of request
           +         catvallen*36      . max. len. of returned value
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen+catvallen . ALS frame size
.
. Code begins here.
.
callsetloc*                            .
           buy       *als_size,*x10,b1 . buy ALS frame
```

```
          lr,u      r1,parlstlen      . param. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set the increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,query_req      . create VA of name
          sa        a2,*4,x10,b1      . set VA of _QUERY in
                                      .    parameter list
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of param. list
          au,u      a1,parlstlen      . create VA of category_value
          sa        a2,*7,x10,b1      . save VA in param. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,3              . number of parameters
          lx,u      x2,setlocva       . load address of setloc
          call      0,x2,b0           . call setloc
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. SETLOCALE was successful.  A1 has length of category_value 7,x10;
. B1 has address of category_value.
.
. Processing of value code goes here.
.
subexit                               .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. SETLOCALE returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                            .
          tn        a0                . error or warning?
          j         processwarn       . warning, process it
                                      . error - setloc failed
.
. Error processing code goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                           .
.
. Warning processing code goes here.
.
```

```
        j       subexit         . exit through common point
        $end                    . end setloc
```

6.33. STRFMON Service Routine

Entry Point

EMSM$STRFMON

Description

STRFMON is used to convert numeric monetary values to a string representation based on a specified format and the locale information defined by the LC_MONETARY category.

Calling Sequence

```
CALL      EMSM$STRFMON
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the STRFMON parameters and the required input:

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the virtual address to the parameter list formatted in the standard calling sequence. These parameters are described as follows:

Parameter 1

is the format string, which describes the format of the resulting output buffer.

Parameter Name: format
Parameter Type: Ordinary Data
Required Access: Read

The format string has two objects: plain characters and conversion specifications.

- Plain Characters: Plain characters are copied unchanged to the output buffer.
- Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

Parameter 2

is the beginning of the list of arguments. For details, go to the Conversion Specification Table.

Parameter Name: arguments
Parameter Type: Ordinary Data
Required Access: Read

Parameter 3

is the output buffer where the resulting string is written. The service routine places characters in this buffer, as controlled by the format string.

Parameter Name: str_out
Parameter Type: Ordinary Data
Required Access: Read, Write

Conversion Specification Table (STRFMON)

The following table shows the sequence of arguments in a conversion specification. Each conversion specification results in the fetching of zero or more sequential arguments that are converted and formatted.

- Each argument must be a double-precision floating point number.
- If excess arguments remain after the format string is exhausted, they are ignored.
- If the format string contains more conversion specifications than arguments, the results are unpredictable. With the C compiler, an error is returned.

Note: *The % character and the conversion character are required; all others are optional.*

Table 6–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
1	% character	Required argument
2	flags	Optional arguments
	=f	A numeric fill character. This character (1) must be representable in a single byte; (2) does not affect field width filling, which always uses the space character; and (3) is ignored unless left precision is specified. The default is the space character.
	^	Indicates that the currency amount should not be formatted with currency characters. If this flag is defined for the current locale, grouping characters are inserted.

Table 6–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
	+ or (	Specifies the style for representing positive and negative currency amounts. Only one style can be specified. If +, the locale's equivalent of + and - are used. If (, negative amounts are enclosed in parentheses ( ). If neither style is specified, the default is +.
	!	Suppresses the currency symbol from output conversion.
	=	Specifies the alignment. If it is present, all fields are left justified.
3	field width (w)	Optional argument. A decimal digit that specifies a minimum field width in bytes. If the - flag is specified, the result is left-justified. If the - flag is not specified, the result is right-justified. The default is zero (0).
4	left precision (#n)	Optional argument. A pound sign followed by a decimal digit string that specifies the maximum number of digits expected to be formatted to the left of the radix character. It can be used to (1) keep formatted output from multiple calls to STRFMON aligned in the same columns and (2) fill unused positions with a special character. • If more than n digit positions are required, this value is ignored, and the excess positions are filled with numeric fill characters (=f). • If the grouping has not been suppressed with the ^ flag and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit. • To ensure alignment, any characters appearing before or after the number in the formatted output (such as currency or sign symbols) are padded as necessary with space characters to make their positive and negative formats an equal length.
5	right precision (.p)	Optional argument. A period followed by a decimal digit that specifies the number of digits after the radix character. If the value is zero, no radix character appears . The amount being formatted is rounded to the specified number of digits before formatting occurs. If right precision is not included, a default specified by the current locale is used.
6	conversion character	Required argument
	i	This argument should use the locale's international currency format. USA example: `USD 1,234.56`
	n	This argument should use the locale's national currency format. USA example: `$1,234.56`
	%	This means no argument is converted. The entire conversion specification must be %%.

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the bit length of the resulting string if A0 indicates normal completion.

Error Statuses

STRFMON can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASREADERR
- 42 – ISLASWRITERR
- 43 – ISLCRTBNKERR
- 44 – ISLCPTRSTERR
- 51 – ISLINVCNVSPC
- 52 – ISLINVARGMTS

Sample Code

This code example shows how to use STRFMON to format a monetary value.

```
        $INCLUDE  'MAXR$/'          .
           $INCLUDE  'OM$DEF'          . OM DEFINITIONS
           $INCLUDE  'CBEP$$ISL-EM'    . I18NLIB ENTRY POINT DEFS
           $EXTEND                     . INDICATE EXTENDED MODE
           $OBJ                        .
           $ASCII                      .
PARDES     $FORM     2,1,1,23,1,1,1,6  . IP,L,U,S & RES.(0),E,R,W,B
.
$(0),DBNK $BANK OM$DATA_EMB 'EMDATA'   .
           $LIT                        . LITERALS TO LOCATION COUNTER 0
```

```
FMT       $EQU      'Cost is %=*#5n (%#5.0n rounded).'
FMTLEN    $EQU      $SL(FMT)           . CHARACTER LENGTH OF STRING
FMTIMAGE  $GEN      FMT                . STRING TO BE FORMATTED
ARGS      +         (12345*-2)D        . ARGUMENTS: 123.45
          +         (12345*-2)D        .            123.45
OUTBUFL   $EQU      10                 . NUMBER OF OUTPUT BUFFER WORDS
OUTBUF    $RES      OUTBUFL            . SPACE FOR RESULTING STRING
.
PARS      PARDES    0,0,0,0,0,1,0,0    . PARAMETER 1 DESCRIPTOR
          +         0                  . VA OF FORMAT STRING
          +         FMTLEN*9           . BIT LENGTH OF STRING
          PARDES    0,0,0,0,0,1,0,0    . PARAMETER 2 DESCRIPTOR
          +         0                  . VA TO START OF ARGUMENTS
          +         0                  . IGNORED
          PARDES    0,0,0,0,0,1,1,0    . PARAMETER 3 DESCRIPTOR
          +         0                  . VA OF OUTPUT BUFFER
          +         OUTBUFL*36         . BIT LENGTH OF BUFFER
PARSLEN   $EQU      $-PARS             . LENGTH OF PARAMETER LIST
.
$(1)      $BANK OM$CODE_EMB 'EMSTART' .
          OM$USE_LV,17 'LV',,B2        . DEFINE LINK VECTOR BANK
START               .
          LBU       B2,R0              . BASE LINK VECTOR BANK ON B2
          OM$LOAD_BDR B3,DBNK          . BASE DATA ON B3
          BUY       *PARSLEN,*X10,B1   . BUY AN ALS FRAME
          LA,U      A1,PARS            . SOURCE ADDRESS (PARAM LIST)
          LXSI,U    A1,1               . INCREMENTOR
          LA        A2,X10             . DESTINATION ON ALS
          LXSI,U    A2,1               . INCREMENTOR
          LR,U      R1,PARSLEN         . NUMBER OF WORDS TO XFER TO ALS
          BT        A2,B1,*A1,B3,*0    . MOVE THE PARAMETER LIST TO ALS
          SBU       B3,A1              . DBANK L,BDI,OFFSET
          AU,U      A1,FMTIMAGE        . VA OF FORMAT STRING IN A2
          SA        A2,*1,X10,B1       . SAVE INTO PARAM ON ALS
          AU,U      A1,ARGS            . VA OF ARGUMENTS LIST IN A2
          SA        A2,*4,X10,B1       . SAVE INTO PARAM ON ALS
          AA,U      A1,OUTBUF          . VA OF OUBPUT BUFFER IN A1
          SA        A1,*7,X10,B1       . SAVE INTO PARAM ON ALS
          SBU       B1,A1              . ALS L,BDI,OFFSET
          AA        A1,X10             . VA TO PARAMETER LIST
          LA,U      A0,3               . NUMBER OF PARAMETERS TO PASS
          DSL       A2,72              . CLEAR REGISTERS A2, A3
          OM$CALL   'EMSM$STRFMON'     . CALL SERVICE ROUTINE
          TE        A0,(045625000000),,B9 . ERROR RETURNED ?
          J         PROCESSERR         . YES
.
. Normal processing
.
          J         COMRTN             . JUMP TO COMMON RETURN
PROCESSERR          .
.
. Error processing
```

```
.
COMRTN              .
          AX,U      X10,PARSLEN       . SELL THE ALS FRAME
               .
               .
               .
```

Assuming that the locale for LC_MONETARY is "POSIX", the following is contained in the output buffer:

```
Cost is $***123.45 ($  123 rounded).
```

6.34. STRFTIME Service Routine

Entry Point

EMSM$STRFTIM

Description

STRFTIME is used to convert date and time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

Calling Sequence

```
CALL      EMSM$STRFTIM
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the STRFTIME parameters and the required input:

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the virtual address to the parameter list formatted in the standard calling sequence. These parameters are described as follows:

Parameter 1

is the format string, which describes the format of the resulting output buffer.

Parameter Name: fmt_str
Parameter Type: Ordinary Data
Required Access: Read

The format string has two objects: plain characters and conversion specifications.

- Plain Characters: Plain characters are copied unchanged to the output buffer.

- Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

Parameter 2

is the date and time structure. To see this structure, go to the Date-and-Time Structures Table. The caller must fill in the required integer values for this structure before making the EMSM$STRFTIM call.

Parameter Name: date_and_time
Parameter Type: Ordinary Data
Required Access: Read

Parameter 3

is the output buffer where the formatted string is placed. The portion of this buffer not occupied by the formatted string is set to binary zeroes by the service routine.

Parameter Name: str_out
Parameter Type: Ordinary Data
Required Access: Read, Write

Conversion Specification Table (STRFTIME)

Each conversion specification (CS) is replaced by the appropriate characters listed in this table. These characters are determined by the current locale and by the values contained in the date-and-time structure. If a CS does not correspond to any of the ones listed in this table, the behavior is undefined.

Table 6–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%a	Weekday name, abbreviated	abday	tm_wday
%A	Weekday name, full	day	tm_wday
%b	Month name, abbreviated	abmon	tm_mon
%B	Month name, full	mon	tm_mon
%c	Appropriate date/time representation	d_t_fmt	Varies by locale
%C	Century number. To get this number, the year is divided by 100 and truncated to an integer as a decimal [00-99].	n/a	tm_year
%d	Day of month as decimal [01,31]	n/a	tm_mday
%D	Equivalent to %m/%d/%y	n/a	tm_mon, tm_mday, and tm_year.

Table 6–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%e	Day of month as decimal [1,31]. A single digit is preceded by a space.	n/a	tm_mday
%F	Equivalent to %Y-%m-%d (the ISO8601:2000 standard date format)	n/a	tm_year, tm_mon, and tm_mday
%g	Replaced by the last 2 digits of the week-based year as a decimal number [00-99].	n/a	tm_year, tm_yday, and tm_mday
%G	Replaced by the week-based year and as a decimal number.	n/a	tm_year, tm_yday, tm_mday
%h	Month name, abbreviated	abmon	tm_mon
%H	Hour on 24-hour clock as decimal [00,23]	n/a	tm_hour
%I	Hour on 12-hour clock as decimal [01,12]	n/a	tm_hour
%j	Day of year as decimal [001,366]	n/a	tm_yday
%m	Month as decimal [01,12]	n/a	tm_mon
%M	Minute as decimal [00,59]	n/a	tm_min
%n	New line character (012)	n/a	n/a
%p	Equivalent for ante-meridiem (AM) and post-meridiem (PM)	am_pm	tm_hour
%r	Time in AM and PM notation; equivalent of %I:%M:%S %p in the POSIX locale	t_fmt_ampm	Refer to individual CSs.
%R	Time in 24-hour notation (%H:%M)	n/a	tm_hour and tm_min.
%S	Second as decimal [00,60]	n/a	tm_sec
%t	Tab character	n/a	n/a
%T	Time (%H:%M:%S)	n/a	tm_hour, tm_min, and tm_sec.
%u	Weekday as decimal [1,7]	n/a	tm_wday
%U	Week number of the year as decimal [00,53] with Sunday as day 1	n/a	tm_yday and tm_wday

Table 6–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%V	Week number of the year as decimal [01,53] with Monday as day 1 If the week containing January 1 has four or more days in the new year, then it is considered week 1. If is does not, then it is the last week of the previous year, and the next week is week 1.	n/a	tm_year, tm_mday, and tm_yday
%w	weekday as decimal [0,6] with 0 representing Sunday	n/a	tm_wday
%W	Week number of the year as decimal [00,53] with Monday as day 1. All days in a new year preceding the first Sunday are considered to be in week 0.	n/a	tm_wday and tm_yday
%x	Appropriate date representation	d_fmt	Varies by locale
%X	Appropriate time representation	t_fmt	Varies by locale
%y	Year without century as decimal [00,99]	n/a	tm_year
%Y	Year with century as decimal [1998]	n/a	tm_year
%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm) or by no characters if no timezone is determinable.	n/a	tm_isdst
%Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.	n/a	tm_isdst
%%	%	n/a	n/a

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the bit length of the formatted string if A0 indicates normal completion.

Error Statuses

STRFTIME can return the following error statuses in Register AO:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASREADERR
- 42 – ISLASWRITERR
- 43 – ISLCRTBNKERR
- 44 – ISLCPTRSTERR
- 51 – ISLINVCNVSPC
- 56 – ISLINVDATTIM
- 59 – ISLSYSTIMERR

Sample Code

This code example shows how to use STRFTIME to format a date value to a string representation based on locale en_US.8859-1 (English as spoken in the US).

```
        $INCLUDE  'MAXR$/'          . DEFINE THE REGISTERS
           $INCLUDE  'OM$DEF'          . DEFINE OM VALUES
           $INCLUDE  'CBEP$$ISL-EM'    . I18NLIB ENTRY DEFINITIONS
           $INCLUDE  'ISL-PKT-DEFS/MSM'.
           $EXTEND                     . INDICATE EXTENDED MODE
           $OBJ                        .
           $ASCII                      .
$(0),DBNK $BANK OM$DATA_EMB 'EMDATA'  .
           $LIT                        . LITERALS TO LOC. COUNTER 0
           $RES      0200              .
STRFTIMVA +         EMSM$STRFTIM      . VA OF STRFTIME
STR       'TODAY IS %x.'              . STRING TO BE FORMATTED
STRLEN    $EQU      $SL('TODAY IS %x.') . CHAR LEN OF STRING
PARDES    $FORM     2,1,1,23,1,1,1,6  . IP,L,U,S & RES.(0),E,R,W,B
PARSKEL             .
           PARDES    0,0,0,0,0,1,0,0   . PARAMETER 1 DESCRIPTOR
```

```
          +         0                 . VIRTUAL ADDR. OF STRING
          +         STRLEN*9          . BIT LENGTH OF STRING
          PARDES    0,0,0,0,0,1,0,0   . PARAMETER 2 DESCRIPTOR
          +         0                 . VIRTUAL ADDR. OF ISLTMPKT
          +         ISLTMLEN*36       . BIT LENGTH OF ISLTMPKT
          PARDES    0,0,0,0,0,1,1,0   . PARAMETER 3 DESCRIPTOR
          +         0                 . VIRTUAL ADDR. OF BUFFER
          +         BUFSIZE*36        . LENGTH OF BUFFER
PARLSTLEN $EQU      $-PARSKEL         . LENGTH OF PARAMETER LIST
ALS_SIZE  $EQU      PARLSTLEN+((STRLEN+3)//4*4)+BUFSIZE . ALS SIZE
BUFSIZE   $EQU      8                 . BUFFER SIZE IN WORDS
TBUF      $RES      BUFSIZE           . # WORDS FOR FORMATTED BUFF
          ISLTMGEN                    . DATE/TIME PKT DEFS & EQUFs
$(1)      $BANK     OM$CODE_EMB 'EMSTART' .
CALLSTRFTIME*       .
          LBU       B2,R0             . BASE LINK VECTOR BANK ON B2
          OM$LOAD_BDR B3,DBNK         . L,BDI OF DATA BANK
          BUY       *ALS_SIZE,*X10,B1 . BUY AN ALS FRAME
          LR,U      R1,PARLSTLEN      . GET LENGTH OF PARAM LIST
          LA,U      A1,PARSKEL        . GET SENDER BANK OFFSET
          LA        A2,X10            . GET RECEIVER BANK OFFSET
          LXSI,U    A1,1              . SET SENDER INCREMENT
          LXSI,U    A2,1              . SET RECEIVER INCREMENT
          BT        A2,B1,*A1,B3,*0   . MOVE PARAM. LIST TO ALS
          SBU       B3,A1             . SAVE L,BDI OF DATA BANK IN A1
          AU,U      A1,STR            . CREATE VA OF FORMAT STR IN A2
          SA        A2,*1,X10,B1      . SAVE STRING VA IN PARAM LIST
          AU,U      A1,TBUF           . CREATE VA OF OUTPUT BUFF IN A2
          SA        A2,*7,X10,B1      . SAVE BUFFER VA IN PARAM LIST
          AA,U      A1,ISLTMPKT       . CREATE VA OF ISLTMPKT
          SA        A1,*4,X10,B1      . SAVE ISLTMPKT VA IN PARAM LIST
          LA,U      A0,ISLTMPKT       . SAVE ISLTMPKT OFFSET IN A0
          LA,U      A1,7              . MONTH OF AUGUST
          SA        A1,ISLTMMON,A0,B3 . SAVE MONTH OF THE YEAR
          LA,U      A1,22             . DAY OF THE MONTH
          SA        A1,ISLTMMDAY,A0,B3 . SAVE DAY OF THE MONTH
          LA,U      A1,95             . YEAR
          SA        A1,ISLTMYEAR,A0,B3 . SAVE YEAR
          SBU       B1,A1             . SAVE L,BDI OF ALS IN A1
          AA        A1,X10            . CREATE VA OF PARAM LIST
          LA,U      A0,3              . GET NUMBER OF PARAMS (3)
          DSL       A2,72             . CLEAR REGISTERS A2, A3
          LX,U      X2,STRFTIMVA      . VA OF STRFTIME
          CALL      0,X2,B3           . TRANSFER CONTROL TO STRFTIME
          LA,U      A5,0,A0           . GET ERROR CODE
          LSSL      A5,18             . CLEAR H1
          SSL       A5,18             .
          JNZ       A5,PROCESSERR     . JUMP TO PROCESS THE ERROR
.
. CONTINUE PROCESSING
.
          J         COMRTN            . JUMP TO COMMON RETURN
```

```
PROCESSERR          .
.
. ERROR PROCESSING
.
COMRTN              .
          AX,U      X10,ALS_SIZE      . SELL THE ALS FRAME
               .
               .
               .
```

Assuming that the locale for LC_TIME is "en_US.8859-1", the following is contained in the output buffer:

```
TODAY IS 08/22/95.
```

6.35. STRING_COLLATE Service Routine

Entry Points

EMSM$STRCOLL

EMSM$STRCOLT

Description

STRING_COLLATE compares two character strings and returns their collating order. These strings can be up to 4096 characters long. The collating rules are determined by the current setting of the LC_COLLATE category.

Use EMSM$STRCOLL as the entry point if you want trailing spaces in input strings treated as significant characters.

Use EMSM$STRCOLT as the entry point if you want trailing spaces truncated.

```
strcollva +       entry-point       . virtual address of STR_COLL
             LA,U    A0,3              . three parameters in list
             LA      A1,virtual-address-of-parameter-list
             LX,U    X2,strcollva      . load address of service routine
             CALL    0,X2,B0           . call the service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following are the STRING_COLLATE parameters and the required input for each:

in-string 1

is the first string containing characters to be compared.

Parameter Type: Ordinary Data
Required Access: Read

out-string 2

is the second string containing characters to be compared.

Parameter Type: Ordinary Data
Required Access: Read

collate-address

the address of a collate table that resides in the application data space. This must be set to zero (0) if no collate table address is to be passed to the service routine.

- If the collate table address is zero (0), STRING_COLLATE finds the collate table based on the value of the category LC_COLLATE.
- If the collate table address is non-zero, STRING_COLLATE assumes that (1) the application has called the GET_COLLATE_TABLE service routine, and (2) the collate table resides in the data space indicated by *collate-address.*

Parameter Type: Ordinary Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the result of the comparison. The possible values are

-1 = the first string collates before the second string.

0 = the strings collate equally.

1 = the first string collates after second string.

If the error status is not OK, this value is undefined.

Error Statuses

STRING_COLLATE can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR

- 26 – ISLLOCNOTFOU
- 28 – ISLINVCAT
- 30 – ISLCORRUPTAB
- 36 – ISLSTR12LONG
- 37 – ISLSTR22LONG
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use STRING_COLLATE to compare two character strings and return their collating order. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                  .
          $lit                        . literals to loc. counter 1
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
strcollva +         EMSM$STRCOLL      . VA of string_collate
instring1 'ABCD'                      .
instringlen1 $equ   $-instring1       . word length of  string 1
instring2 'abcd'                      .
instringlen2 $equ   $-instring2       . word length of string 2
.
. Skeleton of parameter list passed to EMSM$STRCOLL:
.   Parameter 1:  instring1, ordinary data (OD), read access
.   Parameter 2:  instring2, OD, read, write access
.   Parameter 3:  coltbladd, OD, read access
.
parlskel                              .
          pardes    0,0,1,0,0,1       . parameter 1 descriptor
          +         0                 . VA of instring
          +         36*instringlen1   . instring length in bits
          pardes    0,0,1,0,0,1,0,0   . parameter 2 descriptor
          +         0                 . VA of output buffer
          +         36*instringlen2   . output buffer len. in bits
          pardes    0,0,0,0,0,1,0,0   . parameter 3 descriptor
          +         0                 . VA of collate table
          +         0                 . length of collate table
.
. Define length parameters.
.
```

```
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen         . ALS frame size
.
. Skeleton of 2nd parameter list passed to EMSM$STRCOLL:
.   Parameter 1:  instring1, OD, read access
.   Parameter 2:  instring2, OD, read, write access
.   Parameter 3:  coltbladd, OD, read access
.
. Code begins here.
.
callstrcoll*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list len. to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a1             . get L,BDI of code bank
          aa,u      a1,instring1      . create VA of 1st string
          sa        a1,*1,x10,b1      . set instring1 in par. list
          sbu       b0,a1             . get L,BDI of ALS
          aa,u      a1,instring2      . create VA of 2nd string
          sa        a2,*4,x10,b1      . set instring2 in par. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,3              . number of parameters
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of param. list
          lx,u      x2,strcollva      . load VA of string_collate
          call      0,x2,b0           . call string_collate
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. STRING_COLLATE was successful.
.
          jz        a0,collequal      . collate equally?
          te,u      a0,1              . no, collate 2nd str. 1st
          j         firstless         . no-jump
                                      . yes-print ABCD collates 1st
          j         rtn1              . jump to common code
.
collequal                             . print collate equally
          j         rtn1              . jump to common code
firstless                             . print ABCD collates 1st
          j         rtn1              . jump to common code
.
. STRING_COLLATE returned error or warning - process the error.
. A0 still has the error information.
.
```

```
processerr                          . print error msg
.
. Insert error processing here
.
          j         rtn1            . exit through common point
rtn1                                .
          ax,u      x10,als_size    . sell ALS frame
          rtn                       . return to caller
          $end                      . end string_collate
```

6.36. STRING_TRANSFORM Service Routine

Entry Points

EMSM$STRXFRM

EMSM$STRXFRT

Description

STRING_TRANSFORM converts a character string into an ordering key. This string can be up to 4096 characters long. The transformation rules are determined by the current setting of the LC_COLLATE category.

STRING_TRANSFORM: Returning an Ordering Key

STRING_TRANSFORM returns a binary sortable ordering key for the input string. The transformation uses the collation rules for the current locale. The output buffer that is returned from STRING_TRANSFORM is binary 0, which is filled on the right.

Calling Sequence

- Use EMSM$STRXFRM as the entry point if you want trailing spaces in input strings treated as significant characters.
- Use EMSM$STRXFRT as the entry point if you want trailing spaces truncated.

```
strxfrmva +       entry-point       . VA of STRING_TRANSFORM
            LA,U    A0,3              . three parameters in list
            LA      A1,virtual-address-of-parameter-list
            LX,U    X2,strxfrmva      . load address of service routine
            CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following are the STRING_TRANSFORM parameters and the required input for each:

in-buffer

is the buffer containing the characters to be converted.

Parameter Type: Ordinary Data
Required Access: Read

out-buffer

is the buffer into which the service routine places the converted character string.

Parameter Type: Ordinary Data
Required Access: Read and Write

collate-address

the address of a collate table that resides in the application data space. This must be set to zero (0) if no collate table address is to be passed to the service routine.

- If the collate table address is zero (0), STRING_TRANSFORM finds the collate table based on the value of the category LC_COLLATE.
- If the collate table address is non-zero, STRING_TRANSFORM assumes that (1) the application has called the GET_COLLATE_TABLE service routine, and (2) the collate table resides in the data space indicated by *collate-address*.

Parameter Type: Ordinary Data
Required Access: Read

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the bit length of the transformed string.

Error Statuses

STRING_TRANSFORM can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 7 – ISLENCNOTSUP
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 36 – ISLSTR12LONG
- 39 – ISLGETENVERR

- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use STRING_TRANSFORM to convert a character string to an ordering key. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
$(1)                                   .
           $lit                        . literals to loc. counter 1
bufsize    $equ      9                 . buffer size in words
tbuf1      $res      bufsize           . # words for transfrm buffer1
tbuf2      $res      bufsize           . # words for transfrm buffer2
strxfrmva  +         EMSM$STRXFRM      . VA of string_transform
str1       'ABCD'                      . string to be transformed
strlen1    $equ      $SL('ABCD')       . char. length of string1
str2       'abcd'                      . string to be transformed
strlen2    $equ      $SL('abcd')       . char. length of string2
no_bits    $res      1                 . save # of bits from strxfrm
biclds     $form     14,4,1,5,12       . form for BICL descriptors
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
bicldes    biclds    0,a1,0,b0,0       . descriptor word 1 for BICL
           biclds    0,a0,1,b0,0       . descriptor word 2 for BICL
.
. Skeleton of first parameter list passed to EMSM$STRXFRM:
.   Parameter 1: str1, ordinary data (OD), read access
.   Parameter 2: tbuf1, OD, read, write access
.   Parameter 3: coltbladd, OD, read access
.
parlskel1                              .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         36*strlen1        . bit length of string
            pardes   0,0,0,0,0,1,1,0   . parameter 2 descriptor
           +         0                 . VA of buffer
           +         bufsize*36        . bit len. of xfrm buffer
           pardes    0,0,0,0,0,1,0,0   . parameter 3 descriptor
           +         0                 . VA of collate table
           +         0                 . length of collate table
parlstlen1 $equ      $-parlskel1       . length of param. list 1
als_size1 $equ       parlstlen1+((strlen1+3)//4*4)+bufsize . ALS size
.
. Skeleton of second parameter list passed to EMSM$STRXFRM:
.   Parameter 1: str1, OD, read access
```

```
.   Parameter 2: tbuf2, OD, read, write access
.   Parameter 3: coltbladd, OD, read access
.
parlskel2                              .
          pardes    0,0,1,0,0,1,0,0    . parameter 1 descriptor
          +         0                  . VA of string
          +         36*strlen2         . bit length of string
          pardes    0,0,0,0,0,1,1,0    . parameter 2 descriptor
          +         0                  . VA of buffer
          +         bufsize*36         . bit length of buffer
          pardes    0,0,1,0,0,1,0,0    . parameter 3 descriptor
          +         0                  . VA of collate table
          +         0                  . length of collate table
parlstlen2 $equ     $-parlskel2        . length of param. list 2
als_size2 $equ      parlstlen2+((strlen2+3)//4*4)+bufsize . ALS size
.
callstrxfrm*                           .
          buy       *als_size1,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen1      . get length of param. list 1
          la,u      a1,parlskel1       . get sender bank offset par 1
          la        a2,x10             . get receiver bank offset ALS
          lxsi,u    a1,1               . set sndr increment (short)
          lxsi,u    a2,1               . set rcvr increment (short)
          bt        a2,b1,*a1,b0,*0    . move param. list to ALS
          sbu       b0,a1              . save code bank L,BDI in A1
          aa,u      a1,str1            . create VA of string 1
          sa        a1,*1,x10,b1       . save VA in par. list 1
          sbu       b0,a1              . save code bank L,BDI in A1
          aa,u      a1,tbuf1           . create VA of output buffer
          sa        a1,*4,x10,b1       . save VA in par. list 1
          sbu       b1,a1              . save L,BDI of ALS in A1
          aa,u      a1,x10             . create par. list VA for SCS
          la,u      a0,3               . get # of param. (3) for SCS
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72              . clear registers A2, A3
          lx,u      x2,strxfrmva       . get entry point
          call      0,x2,b0            . xfer control to strxfrm
          la,u      a5,0,a0            . get error code only
          jnz       a5,processerr      . jump to process error
.
. First string was successfully transformed.
.
          la,u      a2,no_bits         . get address for no_bits
          sa        a1,0,a2,b0         . save # of bits
          ax,u      x10,als_size1      . sell ALS frame
.
. Transform second string.
.
          buy       *als_size2,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen2      . get len. of param. list 2
```

```
          la,u      a1,parlskel2      . get sender bank offset
                                      .   for parameter 2
          la        a2,x10            . get receiver bank offset ALS
          lxsi,u    a1,1              . set sndr increment (short)
          lxsi,u    a2,1              . set rcvr increment (short)
          bt        a2,b1,*a1,b0,*0   . move param. list to ALS
          sbu       b0,a1             . save code bank L,BDI in A1
          aa,u      a1,str2           . create VA of string
          sa        a1,*1,x10,b1      . set string in param. list
          sbu       b0,a1             . save code bank L,BDI in A1
          aa,u      a1,tbuf2          . create VA of output buffer
          sa        a1,*4,x10,b1      . save VA in ALS
          sbu       b1,a1             . get L,BDI of ALS; save in A1
          aa,u      a1,x10            . create par. list VA for SCS
          la,u      a0,3              . get # of param. (3) for SCS
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          call      strxfrmva,,b0     . call string_transform
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
biclcompa                           .
          la,u      a2,no_bits        . get no_bits addr. of str1
          lr        r1,*0,a2,b0       . get no_bits of str1
          tg        a1,a2             . use largest size to compare
          lr        r1,*0,a1,b0       . get no_bits for str1
          sbu       b0,a1             . save code bank L,BDI in A1
          aa,u      a1,tbuf1          . get VA of transform buffer
          sbu       b0,a0             . save code bank L,BDI in A0
          aa,u      a0,tbuf2          . get VA of transform buffer
          bicl      bicldes,,b0       . compare strings
          jo        eq                . jump if strings are equal
          jc        gt                . if carry, str2 > str1
lt                                    . str2 < str1
.
. Continue processing.
.
          j         comrtn            . jump to common return
gt                                    . str2 > str1;
                                      .   continue processing
          j         comrtn            . jump to common return
eq                                    . strings are equal;
                                      .   continue processing
          j         comrtn            . jump to common return
.
. A0 still has the error information.
.
processerr                            . error processing
          j         comrtn            . jump to common return
comrtn                                .
          ax,u      x10,als_size2     . sell ALS frame
```

```
rtn                             . return to caller
$end                            . end string_transform
```

6.37. STRPTIME Service Routine

Entry Point

EMSM$STRPTIM

Description

STRPTIME is used to convert date and time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

Calling Sequence

```
CALL      EMSM$STRPTIM
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following describes the STRPTIME parameters and the required input:

Parameter 1

is a pointer to the character string to be converted.

Parameter Name: str_in
Parameter Type: Ordinary Data
Required Access: Read

Parameter 2

is a pointer to the format string specified by the conversion specifications.

Parameter Name: fmt_str
Parameter Type: Ordinary Data
Required Access: Read

The format string includes zero or more directives, which consist of one or more white-space characters and either a plain character or a conversion specification.

Directives: The directives are the STRPTIME execution rules. For more details, go to the List of Directives in a Format String.

Plain Character: A plain character is copied unchanged to the output buffer.

Conversion Specification: A conversion specification consists of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

Parameter 3

is a pointer to the date/time structure containing the resultant integer values that are filled in by the service routine.

Parameter Name: date_and_time
Parameter Type: Ordinary Data
Required Access: Read, Write

To see the date/time structure, go to the Date-and-Time Structures Table.

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Error Statuses

STRPTIME can return the following error statuses in Register AO:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASREADERR
- 42 – ISLASWRITERR
- 43 – ISLCRTBNKERR
- 44 – ISLCPTRSTERR
- 51 – ISLINVCNVSPC
- 54 – ISLCHRNOTMCH
- 56 – ISLINVDATTIM

Sample Code

This code example shows how to use STRPTIME to convert date string characters to values based on locale en_US.8859-1 (English as spoken in the US).

```
        $INCLUDE  'MAXR$/'          . DEFINE THE REGISTERS
           $INCLUDE  'OM$DEF'          . DEFINE OM VALUES
           $INCLUDE  'CBEP$$ISL-EM'    . I18NLIB ENTRY DEFINITIONS
           $INCLUDE  'ISL-PKT-DEFS/MSM' .
           $EXTEND                     . INDICATE EXTENDED MODE
           $OBJ                        .
           $ASCII                      .
           OM$USE_LV,17 'LV',,B2       . DEFINE LINK VECTOR BANK
$(0),DBNK $BANK     OM$DATA_EMB 'EMDATA' .
           $LIT                        . LITERALS TO LOCATION COUNTER 0
           $RES      0200              .
STRPTIMVA +         EMSM$STRPTIM      . VA OF STRPTIME
INSTR     'TODAY IS 08/22/95.  '      . STRING TO BE CONVERTED
INSTRLEN  $EQU      $SL('Today is 08/22/95.  ') . CHAR LEN OF INSTR
FORMSTR   'TODAY IS %x.'              . FORMAT STRING
FORMSTRLEN $EQU     $SL('Today is %x.') . CHAR LEN OF FORMAT STRING
PARDES    $FORM     2,1,1,23,1,1,1,6  . IP,L,U,S & RES.(0),E,R,W,B
PARSKEL                               .
           PARDES    0,0,1,0,0,1,0,0   . PARAMETER 1 DESCRIPTOR
           +         0                 . VIRTUAL ADDRESS OF INPUT STRING
           +         INSTRLEN*9        . BIT LENGTH OF INPUT STRING
           PARDES    0,0,1,0,0,1,0,0   . PARAMETER 2 DESCRIPTOR
           +         0                 . VIRTUAL ADDRESS OF FORMAT
STRING
           +         FORMSTRLEN*9      . BIT LENGTH OF FORMAT STRING
           PARDES    0,0,0,0,0,1,0,0   . PARAMETER 3 DESCRIPTOR
           +         0                 . VIRTUAL ADDRESS OF ISLTMPKT
           +         ISLTMLEN*36       . BIT LENGTH OF ISLTMPKT
PARLSTLEN $EQU      $-PARSKEL         . LENGTH OF PARAMETER LIST
ALS_SIZE  $EQU      PARLSTLEN         . ALS SIZE
           ISLTMGEN                    . DATE/TIME PKT DEFS AND EQUFs
$(1)      $BANK     OM$CODE_EMB 'EMSTART' .
CALLSTRPTIME*                         .
           LBU       B2,R0             . BASE LINK VECTOR BANK ON B2
           OM$LOAD_BDR B3,DBNK         . L,BDI OF DATA BANK
           BUY       *ALS_SIZE,*X10,B1 . BUY AN ALS FRAME
           LR,U      R1,PARLSTLEN      . GET LENGTH OF PARAMETER LIST
           LA,U      A1,PARSKEL        . GET SENDER BANK OFFSET
           LA        A2,X10            . GET RECEIVER BANK OFFSET
           LXSI,U    A1,1              . SET SENDER INCREMENT
           LXSI,U    A2,1              . SET RECEIVER INCREMENT
           BT        A2,B1,*A1,B3,*0   . MOVE THE PARAMETER LIST TO ALS
           SBU       B3,A1             . SAVE L,BDI OF DATA BANK IN A1
           AU,U      A1,INSTR          . CREATE VA OF INPUT STR (IN A2)
           SA        A2,*1,X10,B1      . SAVE INPUT STR VA IN PARAM LIST
           AU,U      A1,FORMSTR        . CREATE VA OF FORMAT STR (IN A2)
           SA        A2,*4,X10,B1      . SAVE FORMAT STR VA IN PARAM
```

```
LIST
          AA,U      A1,ISLTMPKT       . CREATE VA OF DATE/TIME PACKET
          SA        A1,*7,X10,B1      . SAVE ISLTMPKT VA IN PARAM LIST
          SBU       B1,A1             . SAVE L,BDI OF ALS IN A1
          AA,U      A1,X10            . CREATE VA OF PARAM LIST
          LA,U      A0,3              . GET NUMBER OF PARAMS (3)
          DSL       A2,72             . CLEAR REGISTERS A2, A3
          LX,U      X2,STRPTIMVA      . VA OF STRPTIME
          CALL      0,X2,B3           . TRANSFER CONTROL TO ROUTINE
          LA,U      A5,0,A0           . GET ERROR CODE
          LSSL      A5,18             . CLEAR H1
          SSL       A5,18             .
          JNZ       A5,PROCESSERR     . JUMP TO PROCESS THE ERROR
.
. CONTINUE PROCESSING
.
          J         COMTRN            . JUMP TO COMMON RETURN
PROCESSERR                            .
.
. ERROR PROCESSING
.
COMRTN                                .
          AX,U      X10,ALS_SIZE      . SELL THE ALS FRAME
               •
               •
               •
```

The significant fields of the date/time structure packet for this locale are filled in as follows:

```
isltmmday   22
isltmmon     7
isltmyear   95
```

6.38. TOLOWER Service Routine

Entry Point

EMSM$TOLOWER

Description

TOLOWER converts characters in a string to their corresponding lowercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

Calling Sequence

```
tolowerva +       EMSM$TOLOWER      . virtual addr. of TOLOWER
          LA,U    A0,2                . two parameters in list
          LA      A1,virtual-address-of-parameter-list
          LX,U    X2,tolowerva      . load address of service routine
          CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following are the TOUPPER and TOLOWER parameters and the required input for each:

in-string

is a pointer to the buffer containing the characters to be converted.

Parameter Type: Ordinary Data
Required Access: Read

out-string

is the buffer into which the service routine places the converted character string.

Parameter Type: Ordinary Data
Required Access: Read and Write

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses..

Register A1

contains the bit length of the converted string.

Error Statuses

TOLOWER can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 17 – ISLOVERLAPOP
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE

- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use TOLOWER to convert an input character string to lowercase alphabetic characters. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                  .
          $lit                        . literals to loc. counter 1
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
          $ascii                      . ASCII characters
pardes    $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
tolowerva +         EMSM$TOLOWER      . VA of tolower
instring  'String to test'            . character string to test
instrlen  $equ      $-instring        . length of instring
outlen    $equ      5                 . limit to 5 words
.
. Skeleton of parameter list passed to EMSM$TOLOWER:
.   Parameter 1:  instring, ordinary data (OD), read access
.   Parameter 2:  outstring, OD, read, write access
.
parlskel                              .
          pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
          +         0                 . VA of string
          +         36*instrlen       . bit length of instring
          pardes    0,0,1,0,0,1,1,0   . parameter 2 descriptor
          +         0                 . VA of string
          +         36*outlen         . bit length of output str
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of parameter list
als_size  $equ      parlstlen+outlen  . ALS frame size
.
. Code begins here.
.
calltolower*                          .
          buy       *als_size,*x10,b1 . buy ALS frame
          lr,u      r1,parlstlen      . par. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,instring       . create VA of the string
          sa        a2,*1,x10,b1      . set string VA in par. list
```

```
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of par. list
          au,u      a1,parlstlen      . create VA of outstring
          sa        a2,*4,x10,b1      . save VA in par. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,2              . number of parameters
          lx,u      x2,tolowerva      . load VA of tolower
          call      0,x2,b0           . call tolower
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. TOLOWER was successful.  Register A1 contains the bit length
. of the converted string located at *parlstlen,x10,b1.
.
subexit                             .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. TOLOWER returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                          . error processing
          tn        a0                . error or warning?
          j         processwarn       . warning, process it
                                      . error-tolower failed
.
. Error processing goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                        .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end tolower
```

6.39. TOUPPER Service Routine

Entry Point

EMSM$TOUPPER

Description

TOUPPER converts characters in a string to their corresponding uppercase representation. The character string must be one or more characters long. This

conversion is based on the processing rules in the locale indicated by the current setting of LC_CTYPE category.

Calling Sequence

```
toupperva  +      EMSM$TOUPPER      . virtual addr. of TOUPPER
              LA,U   A0,2             . two parameters in list
              LA     A1,virtual-address-of-parameter-list
              LX,U   X2,toupperva     . load address of service routine
              CALL   0,X2,B0          . call service routine
```

Parameter List

In the standard calling sequence, the format of each parameter in the list is a three-word entry. The following are the TOUPPER and TOLOWER parameters and the required input for each:

in-string

is a pointer to the buffer containing the characters to be converted.

Parameter Type: Ordinary Data
Required Access: Read

out-string

is the buffer into which the service routine places the converted character string.

Parameter Type: Ordinary Data
Required Access: Read and Write

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses..

Register A1

contains the bit length of the converted string.

Error Statuses

TOUPPER can return the following error statuses in Register A0:

- 0 – ISLOK
- 3 – ISLBUFSHORT
- 15 – ISLTOOMANPAR
- 16 – ISLTOOFEWPAR
- 17 – ISLOVERLAPOP
- 20 – ISLNAMINVCHR

- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 30 – ISLCORRUPTAB
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use TOUPPER to convert an input character string to uppercase alphabetic characters. In this example, it is assumed that MASM produces an extended mode relocatable.

```
        $include  'maxr$/'          . define registers
           $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                   .
           $lit                        . literals to loc. counter 1
           $extend                     . indicate extended mode
           $rel                        . indicate EM relocatable
           $ascii                      . ASCII characters
pardes     $form     2,1,1,23,1,1,1,6  . IP,L,U,S & Res.(0),E,R,W,B
toupperva +         EMSM$TOUPPER      . VA of toupper
instring   'String to test'           . char. string to test
instrlen   $equ      $-instring        . length of instring
outlen     $equ      5                 . limit to 5 words
.
. Skeleton of parameter list passed to EMSM$TOUPPER:
.   Parameter 1:  instring, ordinary data (OD), read access
.   Parameter 2:  outstring, OD, read, write access
.
parlskel                               .
           pardes    0,0,1,0,0,1,0,0   . parameter 1 descriptor
           +         0                 . VA of string
           +         36*instrlen       . bit length of instring
           pardes    0,0,1,0,0,1,1,0   . parameter 2 descriptor
           +          0                . VA of string
           +         36*outlen         . bit length of output str
.
. Define length parameters.
.
parlstlen $equ      $-parlskel        . length of param. list
als_size  $equ      parlstlen+outlen  . ALS frame size
.
. Code begins here.
.
calltoupper*                           .
           buy       *als_size,*x10,b1 . buy ALS frame
```

```
          lr,u      r1,parlstlen      . param. list length to move
          la,u      a1,parlskel       . get sender bank offset
          la        a2,x10            . get receiver bank offset
          lxsi,u    a1,1              . set increments
          lxsi,u    a2,1              .
          bt        a2,b1,*a1,b0,*0   . move parameter list
          sbu       b0,a2             . get L,BDI of code bank
          aa,u      a2,instring       . create VA of the string
          sa        a2,*1,x10,b1      . set string VA in par. list
          sbu       b1,a1             . get L,BDI of ALS
          aa        a1,x10            . create VA of param. list
          au,u      a1,parlstlen      . create VA of outstring
          sa        a2,*4,x10,b1      . save VA in param. list
.
. Clear Registers A2 and A3 for the standard calling sequence.
.
          dsl       a2,72             . clear registers A2, A3
          la,u      a0,2              . number of parameters
          lx,u      x2,toupperva      . load VA of toupper
          call      0,x2,b0           . call toupper
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. TOUPPER was successful.  Register A1 contains the bit length
. of the converted string located at *parlstlen,x10,b1.
.
subexit                             .
          ax,u      x10,als_size      . sell ALS frame
          rtn                         . return to caller
.
. TOUPPER returned an error or warning - process the error.
. A0 still has the error information.
.
processerr                          . error processing
          tn        a0                . error or warning?
          j         processwarn       . warning, process it;
                                      . error-toupper failed
.
. Error processing goes here.
.
          j         subexit           . exit through common point
.
. Process the warning.
.
processwarn                         .
.
. Warning processing code goes here.
.
          j         subexit           . exit through common point
          $end                        . end toupper
```

6.40. TRANSFORM_LENGTH Service Routine

Entry Point

EMSM$TRANLEN

Description

TRANSFORM_LENGTH performs these tasks:

- Calculates the maximum length of the ordering key that STRING_TRANSFORM can create for a given string length.
- Determines the maximum size that will be required for the output buffer when using STRING_TRANSFORM. The size that is returned will depend on the current setting of the LC_COLLATE category. It may be several times larger than the value that was passed.

Calling Sequence

```
tranlenva +       EMSM$TRANLEN      . VA of TRANSFORM_LENGTH
             LA,U    A0,0              . zero parameters in list
             LA,H1   A1,locale-identifier-or-zero
             LA,H2   A1,bit-length-of-string-to-be-transformed
             LX,U    X2,tranlenva      . load address of service routine
             CALL    0,X2,B0           . call service routine
```

Parameter List

In the standard calling sequence, each parameter format is a three-word entry. The following describes the TRANSFORM_LENGTH parameters and the required input for these entries:

Register A1,H1

contains either zero (0) or the numeric identifier for the locale. If *locale identifier* is 0, the locale identified by the LC_COLLATE category is used.

Register A1,H2

contains the bit length of a string that is to be transformed.

Return Values

Register A0

contains the error structure returned by the service routine. See 6.1 for a detailed explanation of this structure and error statuses.

Register A1

contains the maximum bit length required by the STRING_TRANSFORM service routine to hold a transformed string of the length specified by Register A1,H2. If the error status in Register A0 is not OK, the value for Register A1 is undefined.

Error Statuses

TRANSFORM_LENGTH can return the following error statuses in Register A0:

- 0 – ISLOK
- 15 – ISLTOOMANPAR
- 20 – ISLNAMINVCHR
- 21 – ISLVALINVCHR
- 26 – ISLLOCNOTFOU
- 39 – ISLGETENVERR
- 41 – ISLASTRREAD
- 42 – ISLASTRWRITE
- 43 – ISLCREATBANK
- 44 – ISLCHPTRESRT

Sample Code

This code example shows how to use TRANSFORM_LENGTH to determine the length of the output buffer returned by STRING_TRANSFORM for a specific length string. In this example, it is assumed that MASM produces an extended mode relocatable.

```
       $include  'maxr$/'          . define registers
          $include  'cbep$$isl-em'    . def. I18NLIB entry points
$(1)                                  .
          $lit                        . literals to loc. counter 1
          $extend                     . indicate extended mode
          $rel                        . indicate EM relocatable
          $ascii                      . ASCII characters
tranlenva +         EMSM$TRANLEN      . VA of transform_length
.
. Code begins here.
.
CALLTRANLEN*                          .
          l,u       a0,0              . load # of parameters
          l,u       a1,6*9            . string size to be transfrmd
          l,u       x2,tranlenva      . load VA of transform_length
          call      0,x2,b0           . call transform_length
          la,u      a5,0,a0           . get error code only
          jnz       a5,processerr     . jump to process error
.
. TRANSFORM_LENGTH was successful.  Register A1 contains the
. length in bits.
.
subexit                               .
          rtn                         . return to caller
.
. TRANSFORM_LENGTH returned an error or warning - process it.
. A0 still has the error information.
.
```

```
processerr                          . error processing
          tn        a5              . error or warning?
          j         processwarn     . warning, process it
.
. Error - TRANSFORM_LENGTH failed.
.
. Error processing goes here.
.
          j         subexit         . exit through common point
.
. Process the warning.
.
processwarn                         .
.
. Warning processing code goes here.
.
          j         subexit         . exit through common point
          $end                      . end transform_length
```

Section 7
Basic and Extended Mode PLUS

This section describes the I18NLIB service routines available for the Basic Mode and Extended Mode PLUS languages. It also provides the information about the COPY elements that define the structures, statuses, and the service routine procedures used.

7.1. PLUS COPY Elements

I18NLIB provides copy elements installed in SYSLIB*PROC$ that define the service routine procedures as well as the structures and statuses used by these service routines. The following COPY elements are provided along with there contents:

COPY element name	Description
Isl-error-copy/pls	Defines the I18NLIB error structure and error status values. For a detailed description of this element see 7.2.1).
Isl-rou-copy/pls	The Basic Mode PLUS language definition of the service routines, structures, and statuses used by the service routines.
Isl-rou-copy/upls	The Extended Mode PLUS language definition of the service routines, structures, and statuses used by the service routines.

7.2. PLUS Data Structures

This section describes the PLUS data structures unique to the I18NLIB service routines. These structures are defined as DEFINE TYPE definitions in the COPY elements ISL-ROU-COPY/PLS and ISL-ROU-COPY/UPLS.

7.2.1. PLUS Error Structure

The I18N Service Library (I18NLIB) includes the ISL-ERR-COPY/PLS copy element in SYSLIB*PROC$. The following declarations are located in this copy element, the error structure for both Basic and Extended Mode PLUS. The one exception is *s'invalid_bm_address'*, which applies only to Basic Mode PLUS.

For an expanded description of each error, see Appendix A. The error statuses that each service routine can return are listed in the description of the service routine.

```
DEFINE TYPE isl_error_status_def =
      18 BIT STATUS (S'OK'                          : 0,
                     S'INVALID_PACKET_LENGTH'       : 1,
```

```
                        S'INVALID_PACKET_VERSION'      : 2,
                        S'BUFFER_TOO_SHORT'            : 3,
                        S'FROM_CCS_NOT_DEFINED'        : 4,
                        S'TO_CCS_NOT_DEFINED'          : 5,
                        S'ENCODING_NOT_SUPPORTED'      : 7,
                        S'ENCODING_OF_BUFFER_FAILED'   : 8,
                        S'BAD_CHARACTER'               : 13,
                        S'BAD_DESCRIPTOR'              : 14,
                        S'TOO_MANY_PARAMETERS'         : 15,
                        S'TOO_FEW_PARAMETERS'          : 16,
                        S'OVERLAPPING_OPERANDS'        : 17,
                        S'EV_NAME_INVALID_CHAR'        : 20,
                        S'EV_VALUE_INVALID_CHAR'       : 21,
                        S'EV_AREA_FULL'                : 22,
                        S'EV_NAME_ILLEGAL_FOR_TIP'     : 23,
                        S'EV_NAME_NOT_FOUND'           : 24,
                        S'EV_VALUE_TOO_LONG'           : 25,
                        S'LOCALE_NOT_FOUND'            : 26,
                        S'NOT_INTERMEDIATE_LOCALE'     : 27,
                        S'INVALID_CATEGORY'            : 28,
                        S'CATEGORY_NOT_SET'            : 29,
                        S'CORRUPTED_TABLE'             : 30,
                        S'CCS_NOT_FOUND'               : 31,
                        S'FROM_CCS_EQUALS_TO_CCS'      : 32,
                        S'CCS_BANK_FULL'               : 33,
 [basic mode only]      S'INVALID_BM_ADDRESS'          : 34,
                        S'INVALID_EM_ENVIRONMENT'      : 35,
                        S'STRING1_TOO_LONG'            : 36,
                        S'STRING2_TOO_LONG'            : 37,
                        S'PUT_ENV_ERROR'               : 38,
                        S'GET_ENV_ERROR'               : 39,
                        S'CCS_NUMBER_NOT_UNIQUE'       : 40,
                        S'ASTORE_READ_ERROR'           : 41,
                        S'ASTORE_WRITE_ERROR'          : 42,
                        S'CREATE_BANK_ERROR'           : 43,
                        S'CHECKPOINT_RESTART_ERROR'    : 44,
                        S'CCS_NUMBER_FORMAT_UNDEFINED' : 45,
                        S'INVALID_LOCALE_ITEM'         : 50,
                        S'INVALID_CONV_SPEC'           : 51,
                        S'INVALID_ARGUMENTS'           : 52,
                        S'TOO_FEW_ARGUMENTS'           : 53,
                        S'CHARACTERS_NOT_MATCH'        : 54,
                        S'UNSUPPORTED_MODIFIED_CS'     : 55,
                        S'INVALID_DATE_TIME VALUE'     : 56,
                        S'INVALID_NMNB_IDENTIFIER'     : 57,
                        S'INVALID_NMNB_NAME_LENGTH'    : 58,
                        S'SYSTEM_TIME_ERROR'           : 59);

DEFINE TYPE isl_severity_status =
      3 BIT STATUS (S'isl_successful'              : 0,
                    S'isl_informational'           : 2,
                    S'isl_warning'                 : 3,
```

```
                        S'isl_major_error'              : 7);

DEFINE TYPE isl_error_def = MAPPED LOCATABLE
       [ isl_severity                               : isl_severity_status,
         isl_component_id                           : 15 BIT UNSIGNED INTEGER,
         isl_error_status                           : isl_error_status_def
       ! isl_error_num                              : 18 BIT UNSIGNED INTEGER];
```

7.2.2. Date/Time Structures (STRFTIME and STRPTIME)

The date/time structure for these two service routines is a nine-word integer structure. Each field in this structure is described in Table 7–1.

```
DEFINE TYPE isl_date_time_def = MAPPED LOCATABLE
              [ tm_sec   : WORD SIGNED INTEGER RANGE (0 TO 61),
                tm_min   : WORD SIGNED INTEGER RANGE (0 TO 59),
                tm_hour  : WORD SIGNED INTEGER RANGE (0 TO 23),
                tm_mday  : WORD SIGNED INTEGER RANGE (0 TO 31),
                tm_mon   : WORD SIGNED INTEGER RANGE (0 TO 11),
                tm_year  : WORD SIGNED INTEGER,
                tm_wday  : WORD SIGNED INTEGER RANGE (0 TO 6),
                tm_yday  : WORD SIGNED INTEGER RANGE (0 TO 365),
                tm_isdst : WORD STATUS
                                (S'DO_NOT_KNOW'    :  0,
                                 S'NO_DST'         :  1,
                                 S'DST'            :  2)     ];
```

Table 7–1. STRFTIME and STRPTIME Date/Time Structures

This word	Contains or indicates	And must be within this range
tm_sec	Number of seconds after the minute. The upper limit is 60 for handling "leap seconds."	[0, 60]
tm_min	Number of minutes after the hour.	[0, 59]
tm_hour	Number of hours since midnight.	[0, 23]
tm_mday	Day of the month.	[1, 31]
tm_mon	Number of months since January. If the current month is January, this number is 0.	[0, 11]
tm_year	Number of years since 1900 A -1 in this field indicates the year 1899. A 100 in this word indicates the year 2000.	[-1899, 8099]
tm_wday	Number of days since Sunday. If the current day is Sunday, this number is 0.	[0, 6]
tm_yday	Number of days since January 1. If the current day is January 1, this number is 0.	[0,365]

Table 7–1. STRFTIME and STRPTIME Date/Time Structures

This word	Contains or indicates	And must be within this range
tm_isdst	Status of Daylight Savings Time positive, DST in effect; 0, DST not in effect, and negative, no DST information available	language-specific

7.2.3. Name-Number Structure

The name-number structure is used by the NAME_AND_NUMBER service routine to pass and return CCS and locale names and numbers. The description of this packet is:

```
DEFINE TYPE isl_nmnb_information_packet = MAPPED LOCATABLE
            [ isl_nmnb_reserved_1           : 18 BIT UNSIGNED INTEGER,
              isl_nmnb_identifier           : 6 BIT UNSIGNED INTEGER,
              isl_nmnb_char_bit_size        : 6 BIT UNSIGNED INTEGER,
              isl_nmnb_version              : 6 BIT UNSIGNED INTEGER,
              isl_nmnb_number               : WORD SIGNED INTEGER,
              isl_nmnb_ccs_name_length      : WORD SIGNED INTEGER,
              isl_nmnb_ccs_number           : WORD SIGNED INTEGER,
              isl_nmnb_ccs_iso_number       : WORD SIGNED INTEGER,
              isl_nmnb_ccs_sdf_number       : WORD SIGNED INTEGER,
              isl_nmnb_ccs_name             : 16 ASCII CHARACTERS,
              isl_nmnb_locale_name_length   : WORD SIGNED INTEGER,
              isl_nmnb_locale_number        : WORD SIGNED INTEGER,
              isl_nmnb_locale_name          : 16 ASCII CHARACTERS ];

DEFINE isl_nmnb_information_packet_length = 16;
```

7.3. PLUS and UPLS Service Routines

The following subsections describe the PLUS and UPLS bindings for I18NLIB service routines The procedure declaration of these service routines are defined in the in the COPY elements ISL-ROU-COPY/PLS and ISL-ROU-COPY/UPLS.

7.3.1. CCS_NAME_TO_CCS_NUM Service Routine

Entry Points

For Basic Mode PLUS: PLS$CNM2CNB

For Extended Mode PLUS: UPLS$CNM2CNB

Description

CCS_NAME_TO_CCS_NUM transforms a coded character set (CCS) name to its corresponding CCS number.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$cnm2cnb'` to `'upls$cnm2cnb'`.

```
DEFINE TYPE isl_ccs_number_format = WORD STATUS
                                   (S'CCS' : 0,
                                    S'ISO' : 1,
                                    S'SDF' : 2);
PROCEDURE CCS_NAME_TO_CCS_NUM(
                              string_ptr : WORD MACHINE POINTER,
                              string_start_char : WORD SIGNED INTEGER,
                              string_num_of_char : WORD SIGNED INTEGER,
                              ccs_number_format : isl_ccs_number_format,
                             %ccs_number : WORD SIGNED INTEGER)
          IMPORTED('pls$cnm2cnb')
          RESULT  isl_error_def;
```

Parameters

string_ptr

is a pointer to the string containing the CCS name. This name is case-sensitive and must correspond to a CCS_NAME entity in the Repository for ClearPath OS 2200 and a CCS that is active on the system.

string_start_char

is the character position in the string where the first character of the name is located. The position of the first character in the string is 1.

string_num_of_char

is the number of characters in the name.

ccs_number_format

is the kind of CCS number desired.

s'CCS' = the Unisys CCS number for use with CCS_TO_CCS transliteration.

s'ISO' = the ISO CCS number for interoperability use.

s'SDF' = the system data format number for use as an SDF character set type (code type) in SDF control records and data records.

ccs_number

is the CCS number corresponding to the specified name. If the service routine returns an error status other than OK, the value for ccs_number is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

CCS_NAME_TO_CCS_NUM can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 31 – S'CCS_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 45 – S'CCS_NUMBER_FORMAT_UNDEFINED'

Sample Code

This code example shows how to use CCS_NAME_TO_CCS_NUM to transform a CCS name to its corresponding CCS number. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Use CCS_NAME_TO_CCS_NUM to transform a CCS name to its
" corresponding CCS number.
"
" Copy elements containing the declaration of procedure
" CCS_NAME_TO_CCS_NUM, isl_error_def, isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for CCS_NAME_TO_CCS_NUM call:
" return status, string pointer, and character length of
" the name.
"
DECLARE ret_stat        : isl_error_def;
DECLARE ccs_name     : 12 ASCII CHARACTERS LOCATABLE;
DECLARE ccs_number : SIGNED WORD INTEGER LOCATABLE;
"
  ccs_name := 'ISO8859-1';
"
" The following parameters are passed:
"     Pointer to the string
"     Character position of first character in CCS name
"     Number of characters in CCS name
"     CCS number and format
"
  ret_stat :=
```

```
CCS_NAME_TO_CCS_NUM(LOC(ccs_name),1,9,s'SDF',%ccs_number);
"
  IF ret_stat.isl_error_status = S'OK'
    THEN
       .
       .
       .
    ELSE
      IF ret_stat.isl_severity EQ S'isl_warning'
        THEN
          PUT FILE ('SYSPRINT') LIST ('Warning ',
                     ret_stat.isl_error_num, ' Received');
        ELSE
          IF ret_stat.isl_severity NE S'isl_successful'
            THEN
              PUT FILE ('SYSPRINT') LIST ('Error ',
                         ret_stat.isl_error_num, ' Received');
TERM
```

7.3.2. CCS_NUM_TO_CCS_NAME Service Routine

Entry Points

For Basic Mode PLUS: PLS$CNB2CNM

For Extended Mode PLUS: UPLS$CNB2CNM

Description

CCS_NUM_TO_CCS_NAME transforms a coded character set (CCS) number to its corresponding CCS name.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY element to declare this procedure. For Extended Mode PLUS, change `'pls$cnb2cnm'` to `'upls$cnb2cnm'`.

```
DEFINE TYPE isl_ccs_number_format = WORD STATUS
                                    (S'CCS' : 0,
                                     S'ISO' : 1,
                                     S'SDF' : 2);
PROCEDURE CCS_NUM_TO_CCS_NAME(
                              ccs_number_format : isl_ccs_number_format,
                              ccs_number : WORD SIGNED INTEGER,
                              string_ptr : WORD MACHINE POINTER,
                              string_start_char : WORD SIGNED INTEGER,
                             %string_num_of_char : WORD SIGNED INTEGER)
          IMPORTED('pls$cnb2cnm')
          RESULT  isl_error_def;
```

Parameters

ccs_number_format

is the kind of CCS number provided:

s'CCS' = the Unisys CCS number for use with CCS_TO_CCS transliteration

s'ISO' = the ISO CCS number for interoperability use

s'SDF' = the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records

ccs_number

is the CCS number. The CCS must participate in a CCS-TO-CCS conversion that is active on the system.

string_ptr

is a pointer to the string where the service routine should write the corresponding CCS name. This name is case-sensitive and corresponds to a CCS_NAME entity in the Repository for ClearPath OS 2200. If the service routine returns an error status other than OK, the value for the CCS name is undefined.

string_start_char

is the character position in the string where the first character should be written. The position of the first character in the string is 1.

string_num_of_char

Upon entry to the service routine, this integer should be set to the character length of the container pointed to by string_ptr.

If the service routine succeeds, it sets string_num_of_char to the number of characters in the locale name.

If the service routine fails, the length is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

CCS_NUM_TO_CCS_NAME can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'

- 3 – S'BUFFER_TOO_SHORT'
- 31 – S'CCS_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 40 – S'CCS_NUMBER_NOT_UNIQUE'
- 45 – S'CCS_NUMBER_FORMAT_UNDEFINED'

Sample Code

This code example shows how to use CCS_NUM_TO_CCS_NAME to transform a CCS number to its corresponding CCS name. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Use CCS_NUM_TO_CCS_NAME to transform a CCS number to its
" corresponding CCS name.
"
" Copy elements containing the declaration of procedure
" CCS_NUM_TO_CCS_NAME, isl_error_def, and isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for CCS_NUM_TO_CCS_NAME call:
" return status, string pointer, and character length of
" output buffer.
"
DECLARE ret_stat           : isl_error_def;
DECLARE ccs_name        : 16 ASCII CHARACTERS LOCATABLE;
DECLARE ccs_number    : SIGNED WORD INTEGER LOCATABLE;
DECLARE ccs_name_len : SIGNED WORD INTEGER LOCATABLE;
"
  ccs_number := 35;
  ccs_name_len := 16;
"
" The following parameters are passed:
"     CCS number and format
"     Pointer to CCS name
"     Character position of first character in CCS name
"     Number of characters in CCS name
"
  ret_stat := CCS_NUM_TO_CCS_NAME(s'SDF',ccs_number,LOC(ccs_name),1,
              %ccs_name_len);
"
  IF ret_stat.isl_error_status = S'OK'
    THEN
        .
        .
        .
```

```
    ELSE
      IF ret_stat.isl_severity EQ S'isl_warning'
        THEN
          PUT FILE ('SYSPRINT') LIST ('Warning ',
                     ret_stat.isl_error_num, ' Received');
        ELSE
          IF ret_stat.isl_severity NE S'isl_successful'
            THEN
              PUT FILE ('SYSPRINT') LIST ('Error ',
                         ret_stat.isl_error_num, ' Received');
TERM
```

7.3.3. CCS_TO_CCS Service Routine

Entry Points

For Basic Mode PLUS: PLS$CCS2CCS

For Extended Mode PLUS: UPLS$CCS2CCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another. The CCS_TO_CCS service routine performs the actual character string transliteration.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$ccs2ccs'` to `'upls$ccs2ccs'`.

```
PROCEDURE ccs_to_ccs(
                    cv_descriptor : WORD SIGNED INTEGER,
                    from_string_ptr : WORD MACHINE POINTER,
                    from_start : WORD SIGNED INTEGER,
                    from_length : WORD SIGNED INTEGER,
                    to_string_ptr : WORD MACHINE POINTER,
                    to_start : WORD SIGNED INTEGER,
                    %to_length : WORD SIGNED INTEGER)
          IMPORTED('PLS$CCS2CCS')
          RESULT  isl_error_def;
```

Parameters

cv_descriptor

is the conversion descriptor (internal identifier for the transliteration tables) returned by OPEN_CCS_TO_CCS.

from_string_ptr

is a pointer to the string containing the characters to be transliterated.

from_start

is the character position in the string where the first character to be transliterated will be located. The first position in the string is 1.

from_length

is the number of bytes to be converted, beginning at from_start. If from_ccs in the OPEN_CCS_TO_CCS call was Fieldata, from_length is the number of Fieldata characters to be converted.

to_string_ptr

is a pointer to the buffer in which CCS_TO_CCS places the transliterated string.

to_start

is the character position in the string where the transliterated string is to begin. The first position in the string is 1.

to_length

is the byte length of the transliterated string set by CCS_TO_CCS. Before calling the service routine, the caller must set this parameter to the number of bytes available in the buffer for storing the transliterated characters.

RESULT

is the error status returned by the service routine.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

The string transliteration service routines can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 4 – S'FROM_CCS_NOT_DEFINED'
- 5 – S'TO_CCS_NOT_DEFINED'
- 13 – S'BAD_CHARACTER'

- 14 – S'BAD_DESCRIPTOR'
- 17 – S'OVERLAPPING_OPERANDS'
- 30 – S'CORRUPTED_TABLE'
- 31 – S'CSS_NOT_FOUND'
- 32 – S'FROM_CCS_EQUALS_TO_CCS'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'

Sample Code

This code example shows how to use OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS to transliterate a character string from one CCS to another. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure
" CCS_TO_CCS.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the CCS_TO_CCS call.
"
DEFINE C646DE_CCS_NUMBER = 20;
DEFINE C8859_1_CCS_NUMBER = 35;
DECLARE ret_stat         : isl_error_def;
DECLARE descriptor     : WORD SIGNED INTEGER;
DECLARE from_string  : 4 CHARACTERS LOCATABLE;
DECLARE to_buffer      : 4 CHARACTER LOCATABLE;
DECLARE to_length      : WORD SIGNED INTEGER LOCATABLE;
"
" This example illustrates procedures to transliterate the
" characters of the German word sweet (süß) in ISO 646DE
" to ISO 8859.1.
"
" First, OPEN_CCS_TO_CCS is called to obtain the conversion
" descriptor.
"
" Next, CCS_TO_CCS is called by using the conversion
" descriptor, along with the string of 'süß', starting position,
" the length of string, etc., as parameters.  CCS_TO_CCS
" places the transliterated string in the buffer pointed by
" to_buffer.
"
" Finally, CLOSE_CCS_TO_CCS is called to complete the
" transliteration.
"
```

```
  descriptor := open_ccs_to_ccs(C646DE_CCS_NUMBER,
                  C8859_1_CCS_NUMBER,%ret_stat);
"
  IF ret_stat.isl_error_status NE S'OK' THEN
     PUT FILE ('SYSPRINT') LIST(ret_stat.isl_error_num);
    ELSE
      BEGIN
      from_string := 'süß';
      to_length := 4;
      ret_stat := ccs_to_ccs(descriptor,LOC(from_string),1,3,
                  LOC(to_buffer),1,%to_length);
"
      IF ret_stat.isl_error_status NE S'OK' THEN
         PUT FILE ('SYSPRINT') LIST(ret_stat.isl_error_num);
      ret_stat := close_ccs_to_ccs(descriptor);
      END;
       .
       .
       .
TERM
```

7.3.4. CLOSE_CCS_TO_CCS Service Routine

Entry Points

For Basic Mode PLUS: PLS$CLOSCCS

For Extended Mode PLUS: UPLS$CLOSCCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$closccs'` to `'upls$closccs'`.

```
PROCEDURE close_ccs_to_ccs(
                          cv_descriptor : WORD SIGNED INTEGER)
          IMPORTED('PLS$CLOSCCS')
          RESULT  isl_error_def;
```

Parameters

cv_descriptor

is the conversion descriptor (internal identifier for the transliteration tables) returned by OPEN_CCS_TO_CCS.

RESULT

is the error status returned by the service routine.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

See 7.3.3 for the error statuses.

Sample Code

See 7.3.3 for sample code.

7.3.5. CONVERT_FORM Service Routine

Entry Points

For Basic Mode PLUS: PLS$CNVRTFM

For Extended Mode PLUS: UPLS$CNVRTFM

Description

CONVERT_FORM performs these tasks:

- Converts a character from its current representation in one portion of a character set to its representation in another portion of the same character set. There is always a one-to-one correspondence in the transliteration that occurs.
- Uses the current value of the locale in the LC_CTYPE category to determine which transliteration table to use. The calling program must ensure that LC_CTYPE is set to a locale that contains the appropriate transliteration table.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure For Extended Mode PLUS, change `'pls$cnvrtfm'` to `'upls$cnvrtfm'`.

```
PROCEDURE convert_form(
                      string_in_ptr  : WORD MACHINE POINTER,
                      start_char_in  : WORD SIGNED INTEGER,
                      num_of_char_in : WORD SIGNED INTEGER,
                      string_out_ptr : WORD MACHINE POINTER,
                      start_char_out : WORD SIGNED INTEGER,
                      convert_dir    : WORD STATUS (S'TO_7_BIT' :7,
                                                    S'TO_8_BIT' :8))
          IMPORTED('pls$cnvrtfm')
          RESULT  isl_error_def;
```

Parameters

string_in_ptr

is a pointer to the string containing the characters to be transliterated.

start_char_in

is the character position in the string of the first character to be transliterated. The position of the first character in the string is 1.

num_of_char_in

is the number of characters to be converted, beginning at start_char_in.

string_out_ptr

is a pointer to the buffer in which the service routine places the converted character string. The output buffer can be the same as the input buffer; it must be at least as large as the input string.

start_char_out

is the character position in the buffer at which the transliterated string is to begin. The position of the first character in the string is 1.

convert_dir

is a status variable that indicates whether the characters are to be converted to the 7-bit range or to the 8-bit range.

RESULT

is the error structure (see 7.2) returned by the service routine.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

CONVERT_FORM can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 17 – S'OVERLAPPING_OPERANDS
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 27 – S'NOT_INTERMEDIATE_LOCALE'
- 30 – S'CORRUPTED_TABLE'

- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code routine shows how to use CONVERT_FORM to change the encoding of a character string. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Use CONVERT_FORM to change the encoding of a character
" string.
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure
" CONVERT_FORM.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the CONVERT_FORM call.
"
DECLARE ret_stat          : isl_error_def;
DECLARE strng             : 20 CHARACTERS LOCATABLE;
DECLARE outbuff           : 20 CHARACTERS LOCATABLE;
DECLARE num_char_in  : WORD SIGNED INTEGER;
"
" In this example, the German word 'süß' is being converted.
" The internal octal representation of 'süß' in ISO 646DE is
"   s = 163
"   ü = 175
"   ß = 176
"
   strng := 'süß';
"
" Call CONVERT_FORM to convert the string to its 8-bit
" representation in ISO 8859-1.  The locale that will be
" used for the transliteration will be determined by the
" setting of LC_CTYPE.  This locale must be defined as a
" Unisys intermediate locale for Germany.
"
   ret_stat:= convert_form(LOC(strng),1,3,LOC(outbuff),
      1,S'TO_8_BIT');
"
```

```
   IF ret_stat.isl_error_status NE S'OK'
     THEN
       BEGIN
         .                          " conversion in error
         .
         .
       END;
"
" The output buffer will contain the internal octal
" representation of 'süß' in ISO 8859-1.
"   s = 163
"   ü = 374
"   ß = 337
TERM
```

7.3.6. GETENV Service Routine

Entry Points

For Basic Mode PLUS: PLS$GETENV

For Extended Mode PLUS: UPLS$GETENV

Description

GETENV retrieves the current value of the environment variable set at the run level. These variables are initially set by the default values from the user profile.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isalnum'` to `'upls$isalnum'`.

```
PROCEDURE getenv(
                env_var_name_ptr      : WORD MACHINE POINTER,
                env_var_name_length   : WORD SIGNED INTEGER,
                env_var_value_ptr     : WORD MACHINE POINTER,
                %env_var_value_length : WORD SIGNED INTEGER
                                      : (LOCATABLE))
         IMPORTED('pls$getenv')
         RESULT  isl_error_def;
```

Parameters

env_var_name_ptr

is a pointer to a string containing the environment variable name.

env_var_name_length

is the character length of the environment variable name.

env_var_value_ptr

is a pointer to a string in which the service routine places the environment variable value. If the operation fails, this value is undefined.

env_var_value_length

is an integer that is set by the service routine to the character length of the environment variable value. Upon entry to the service routine, this integer is equal to the character length of the container holding the value. If the operation fails, this length is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

GETENV can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_ CHAR'
- 24 – S'EV_NAME_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT
- 39 – S'GET_ENV_ERROR'

Sample Code

This code example shows how to use GETENV to retrieve the value of environment variables (EV) at the run level. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`

```
MODULE;
"
"Copy elements containing the declaration of procedure
" GETENV, isl_error_def, and isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for the GETENV call, the return status,
" the EV name and value, and the character length of the
" EV value.
"
DECLARE ret_stat            : isl_error_def;
```

```
DECLARE env_var             : 20 CHARACTERS LOCATABLE;
DECLARE env_val             : 16 CHARACTERS LOCATABLE;
DECLARE env_val_length      : SIGNED WORD INTEGER LOCATABLE;
"
" The EV name is LC_ALL.
"
  env_var := 'LC_ALL';
"
" The EV value length is set to 16.
"
  env_val_length := BYTESIZE(env_val);
"
" The following parameters are passed:
"     Pointer to EV name
"     EV name length=6
"     Pointer to EV value
"     EV value length passed by reference
"
  ret_stat := getenv(LOC(env_var),6,LOC(env_val),%env_val_length);
"
" Display EV value.
"
  IF ret_stat.isl_error_status = S'OK'
    THEN
        PUT FILE ('SYSPRINT') EDIT ('Current LC_ALL value is ',
        env_val(1 TO env_val_length))(A(24),A,SKIP);
TERM
```

7.3.7. GETENVSYS Service Routine

Entry Points

For Basic Mode PLUS: PLS$GETENVS

For Extended Mode PLUS: UPLS$GETENVS

Description

GETENVSYS retrieves the current value of the environment variable set at the system level.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isalnum'` to `'upls$isalnum'`.

```
PROCEDURE getenvs(
                  env_var_name_ptr : WORD MACHINE POINTER,
                  env_var_name_length : WORD SIGNED INTEGER,
                  env_var_value_ptr : WORD MACHINE POINTER,
                  %env_var_value_length : WORD SIGNED INTEGER
                                        : (LOCATABLE))
          IMPORTED('pls$getenvs')
          RESULT  isl_error_def;
```

Parameters

env_var_name_ptr

is a pointer to a string containing the environment variable name.

env_var_name_length

is the character length of the environment variable name.

env_var_value_ptr

is a pointer to a string in which the service routine places the environment variable value. If the operation fails, this value is undefined.

env_var_value_length

is an integer that is set by the service routine to the character length of the environment variable value. Upon entry to the service routine, this integer is equal to the character length of the container holding the value. If the operation fails, this length is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

GETENVSYS can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'

- 24 – S'EV_NAME_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT
- 39 – S'GET_ENV_ERROR'

Sample Code

This code example shows how to use GETENVSYS to retrieve the value of environment variables (EV) at the system level. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure
" GETENVSYS, isl_error_def, and isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for the GETENVSYS call, the return status,
" the EV name and EV value, and the character length of the
" EV value.
"
DECLARE ret_stat       : isl_error_def;
DECLARE env_var        : 20 CHARACTERS LOCATABLE;
DECLARE env_val        : 16 CHARACTERS LOCATABLE;
DECLARE env_val_length : SIGNED WORD INTEGER LOCATABLE;
"
" EV name is LC_ALL.
"
  env_var := 'LC_ALL';
"
" Length of EV value is set to 16.
"
  env_val_length := BYTESIZE(env_val);
"
" Parameters passed are:
"     Pointer to EV name
"     EV name length=6
"     Pointer to EV value
"     EV value length passed by reference
"
  ret_stat := getenvsys(LOC(env_var),6,LOC(env_val),%env_val_length);
"
" Display EV value.
"
  IF ret_stat.isl_error_status = S'OK'
    THEN
        PUT FILE ('SYSPRINT') EDIT ('Current LC_ALL value is ',
        env_val(1 TO env_val_length))(A(24),A,SKIP);
TERM
```

7.3.8. ISALNUM Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISALNUM

For Extended Mode PLUS: UPLS$ISALNUM

Description

ISALNUM determines if all the characters in the string are in the alphanumeric character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the alpha and digit categories in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isalnum'` to `'upls$isalnum'`.

```
PROCEDURE isalnum(
                  string_ptr : WORD MACHINE POINTER,
                  start_char : WORD SIGNED INTEGER,
                  number_of_characters : WORD SIGNED INTEGER,
                 %error_status : isl_error_def)
          IMPORTED('pls$isalnum')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISALNUM to determine if all characters of the input string are alphanumeric. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned.
" Copy element containing the declaration of procedure ISALNUM.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISALNUM call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
```

```
" Call ISALNUM with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF isalnum(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .                          " string is alphanumeric
         .
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
             .                      " string is not alphanumeric
             .
             .
           END;
         ELSE
           BEGIN
             .                      " an error occurred
             .
             .
           END;
        END;
         .
         .
         .
TERM
```

7.3.9. ISALPHA Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISALPHA

For Extended Mode PLUS: UPLS$ISALPHA

Description

ISALPHA determines if all the characters in the string are in the alphabetic character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the upper, lower, and letter categories in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isalpha'` to `'upls$isalpha'`.

```
PROCEDURE isalpha(
                  string_ptr : WORD MACHINE POINTER,
                  start_char : WORD SIGNED INTEGER,
                  number_of_characters : WORD SIGNED INTEGER,
                 %error_status : isl_error_def)
          IMPORTED('pls$isalpha')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'

- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISALPHA to determine if all characters of the input string are alphabetic. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISALPHA call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
" Call ISALPHA with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF isalpha(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                          " string is alphabetic
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
             .
             .                      " string is not alphabetic
             .
```

```
          END;
        ELSE
          BEGIN
            .
            .                          " an error occurred
            .
          END;
END;
 .
 .
 .
TERM
```

7.3.10. ISCNTRL Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISCNTRL

For Extended Mode PLUS: UPLS$ISCNTRL

Description

ISCNTRL determines if all the characters in the string are in the control character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the cntrl category in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$iscntrl'` to `'upls$iscntrl'`.

```
PROCEDURE iscntrl(
                string_ptr : WORD MACHINE POINTER,
                start_char : WORD SIGNED INTEGER,
                number_of_characters : WORD SIGNED INTEGER,
               %error_status : isl_error_def)
          IMPORTED('pls$iscntrl')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISCNTRL to determine if all characters of the input string are control characters. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure ISCNTRL.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISCNTRL call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
" Call ISCNTRL with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF iscntrl(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .                        " string is all control
         .                              characters
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
             .                    " string is not all control
             .                           characters
             .
           END;
         ELSE
           BEGIN
             .                    " an error occurred
             .
             .
           END;
     END;
   .
   .
   .
TERM
```

7.3.11. ISDIGIT Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISDIGIT

For Extended Mode PLUS: UPLS$ISDIGIT

Description

ISDIGIT determines if all the characters in the string are decimal digits in the numeric character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the digit category in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isdigit'` to `'upls$isdigit'`.

```
PROCEDURE isdigit(
                  string_ptr : WORD MACHINE POINTER,
                  start_char : WORD SIGNED INTEGER,
                  number_of_characters : WORD SIGNED INTEGER,
                 %error_status : isl_error_def)
          IMPORTED('pls$isdigit')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISDIGIT to determine if all the characters of the input string are numeric (decimal digits). For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure ISDIGIT.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISDIGIT call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
```

```
" Call ISDIGIT with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF isdigit(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                        " string is all numeric characters
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
             .
             .                    " string is not all numeric
characters
             .
           END;
         ELSE
           BEGIN
             .
             .                    " an error occurred
             .
           END;
       END;
        .
        .
        .
TERM
```

7.3.12. ISGRAPH Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISGRAPH

For Extended Mode PLUS: UPLS$ISGRAPH

Description

ISGRAPH determines if all the characters in the string are in the graphic character class.

This operation is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the graph category in locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isgraph'` to `'upls$isgraph'`.

```
PROCEDURE isgraph(
                  string_ptr : WORD MACHINE POINTER,
                  start_char : WORD SIGNED INTEGER,
                  number_of_characters : WORD SIGNED INTEGER,
                 %error_status : isl_error_def)
          IMPORTED('pls$isgraph')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'

- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISGRAPH to determine if all the characters of the input string are graphic characters. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure ISGRAPH.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISGRAPH call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
" Call ISGRAPH with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF isgraph(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                        " string is all graphic characters
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
```

```
              .
              .                    " string is not all graphic
  characters
              .
            END;
          ELSE
            BEGIN
              .
              .                    " an error occurred
              .
            END;
  TERM
```

7.3.13. ISLOWER Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISLOWER

For Extended Mode PLUS: UPLS$ISLOWER

Description

ISLOWER determines if all the characters in the string are in the lowercase character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the lower category in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$islower'` to `'upls$islower'`.

```
PROCEDURE islower(
                string_ptr : WORD MACHINE POINTER,
                start_char : WORD SIGNED INTEGER,
                number_of_characters : WORD SIGNED INTEGER,
               %error_status : isl_error_def)
          IMPORTED('pls$islower')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISLOWER to determine if the characters of the input string are lowercase and alphabetic. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure ISLOWER.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISLOWER call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
" Call ISLOWER with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF islower(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                    " string is all lowercase characters
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
             .
             .                " string is not all lowercase characters
             .
           END;
         ELSE
           BEGIN
             .
             .                " an error occurred
             .
           END;
       END;
        .
        .
        .
TERM
```

7.3.14. ISPRINT Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISPRINT

For Extended Mode PLUS: UPLS$ISPRINT

Description

ISPRINT determines if all the characters in the string are in the printable character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the print category in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isprint'` to `'upls$isprint'`.

```
PROCEDURE isprint(
                  string_ptr : WORD MACHINE POINTER,
                  start_char : WORD SIGNED INTEGER,
                  number_of_characters : WORD SIGNED INTEGER,
                 %error_status : isl_error_def)
          IMPORTED('pls$isprint')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISPRINT to determine if the characters of the input string are printable. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned, and
" Copy element containing the declaration of procedure ISPRINT.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISPRINT call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
```

```
" Call ISPRINT with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF isprint(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                    " string is all printable characters
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
             .
             .                " string is not all printable characters
             .
           END;
         ELSE
           BEGIN
             .
             .                " an error occurred
             .
           END;
       END;
        .
        .
        .
TERM
```

7.3.15. ISPUNCT Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISPUNCT

For Extended Mode PLUS: UPLS$ISPUNCT

Description

ISPUNCT determines if all the characters in the string are in the punctuation character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the punct category in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$ispunct'` to `'upls$ispunct'`.

```
PROCEDURE ispunct(
                  string_ptr : WORD MACHINE POINTER,
                  start_char : WORD SIGNED INTEGER,
                  number_of_characters : WORD SIGNED INTEGER,
                 %error_status : isl_error_def)
          IMPORTED('pls$ispunct')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'

- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISPUNCT to determine if the characters of the input string are punctuation characters. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure ISPUNCT.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISPUNCT call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
" Call ISPUNCT with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF ispunct(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                  " string is all punctuation characters
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
```

```
                .
                .               " string is not all punctuation characters
                .
              END;
            ELSE
              BEGIN
                .
                .               " an error occurred
                .
              END;
          END;
           .
           .
           .
TERM
```

7.3.16. ISSPACE Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISSPACE

For Extended Mode PLUS: UPLS$ISSPACE

Description

ISSPACE determines if all the characters in the string are in the white space character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the space category in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isspace'` to `'upls$isspace'`.

```
PROCEDURE isspace(
                string_ptr : WORD MACHINE POINTER,
                start_char : WORD SIGNED INTEGER,
                number_of_characters : WORD SIGNED INTEGER,
               %error_status : isl_error_def)
          IMPORTED('pls$isspace')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISSPACE to determine if the characters of the input string are white space characters. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure ISSPACE.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISSPACE call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
" Call ISSPACE with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF isspace(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                      " string is all space characters
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
             .
             .                  " string is not all space characters
             .
           END;
         ELSE
           BEGIN
             .
             .                  " an error occurred
             .
           END;
       END;
         .
         .
         .
TERM
```

7.3.17. ISUPPER Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISUPPER

For Extended Mode PLUS: UPLS$ISUPPER

Description

ISUPPER determines if all the characters in the string are in the uppercase character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the upper category in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isupper'` to `'upls$isupper'`.

```
PROCEDURE isupper(
                  string_ptr : WORD MACHINE POINTER,
                  start_char : WORD SIGNED INTEGER,
                  number_of_characters : WORD SIGNED INTEGER,
                 %error_status : isl_error_def)
          IMPORTED('pls$isupper')
          RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISUPPER to determine if the characters of the input string are uppercase and alphabetic. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure ISUPPER.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISUPPER call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
```

```
" Call ISUPPER with the following parameters:
"    Pointer to string to be tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF isupper(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                    " string is all uppercase characters
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
             .
             .                " string is not all uppercase characters
             .
           END;
         ELSE
           BEGIN
             .
             .                " an error occurred
             .
           END;
       END;
       .
       .
       .
TERM
```

7.3.18. ISXDIGIT Service Routine

Entry Points

For Basic Mode PLUS: PLS$ISXDIG

For Extended Mode PLUS: UPLS$ISXDIG

Description

ISXDIGIT determines if all the characters in the string are in the hexadecimal digit (xdigit) character class.

This operation is based on the processing rules for the locale indicated by the current setting of the LC_CTYPE category.

The allowable values for a locale are determined by the setting of the xdigit category in the locale definition file.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isxdig'` to `'upls$isxdig'`.

```
PROCEDURE isxdigit(
                string_ptr : WORD MACHINE POINTER,
                start_char : WORD SIGNED INTEGER,
                number_of_characters : WORD SIGNED INTEGER,
               %error_status : isl_error_def)
        IMPORTED('pls$isxdig')
        RESULT CONDITION;
```

Parameters

The following variables and their values apply to all the character class service routines.

string_ptr

is a pointer to the characters to be tested.

start_char

is the position in the string where the service routine is to begin. The first character in the string is 1.

number_of_characters

is the number of characters to be tested, beginning at start_char.

error_status

is the error structure returned by the service routine (see 7.2).

RESULT

If the characters are the correct character class for the service routine, the value is TRUE.

If one or more characters are not the correct character class or an error occurred, the value is FALSE. If FALSE is returned, check *error_status* to determine if an error occurred.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

These are the error statuses that can be returned in the *isl-error-def* record by the character class service routines.

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'

- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use ISXDIGIT to determine if the characters of the input string are hexadecimal digits. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure ISXDIGIT.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the ISXDIGIT call.
"
DECLARE ret_stat : isl_error_def;
DECLARE a_string : 20 CHARACTERS LOCATABLE;
"
" Call ISXDIGIT with the following parameters:
"    Pointer to string to tested
"    Position where service routine begins checking
"    Number of characters to check
"    Error status returned
"
   IF isxdigit(LOC(a_string),6,12,%ret_stat)
     THEN
       BEGIN
         .
         .                    " string is all hexadecimal digits
         .
       END;
     ELSE
       BEGIN
       IF isl_error_status EQ S'OK'
         THEN
           BEGIN
```

```
                    .
                    .                " string is not all hexadecimal digits
                    .
                  END;
                ELSE
                  BEGIN
                    .
                    .                " an error occurred
                    .
                  END;
              END;
               .
               .
               .
        TERM
```

7.3.19. LOCALE_NAME_TO_CCS_NUM Service Routine

Entry Points

For Basic Mode PLUS: PLS$LNM2CNB

For Extended Mode PLUS: UPLS$LNM2CNB

Description

LOCALE_NAME_TO_CCS_NUM returns the coded character set (CCS) number associated with the locale name.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$lnm2cnb'` to `'upls$lnm2cnb'`.

```
DEFINE TYPE isl_ccs_number_format = WORD STATUS
                                    (S'CCS' : 0,
                                     S'ISO' : 1,
                                     S'SDF' : 2);
PROCEDURE LOCALE_NAME_TO_CCS_NUM(
                                  string_ptr : WORD MACHINE POINTER,
                                  string_start_char : WORD SIGNED INTEGER,
                                  string_num_of_char : WORD SIGNED INTEGER,
                                  ccs_number_format : isl_ccs_number_format,
                                 %ccs_number : WORD SIGNED INTEGER)
          IMPORTED('pls$lnm2cnb')
          RESULT  isl_error_def;
```

Parameters

string_ptr

is a pointer to the string containing the locale name. This name is required for the SETLOCALE service routine. It is case-sensitive and must correspond to a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and a locale that is active on the system.

string_start_char

is the character position in the string where the first character of the name is located. The position of the first character in the string is 1.

string_num_of_char

is the number of characters in the name.

ccs_number_format

is the kind of CCS number desired:

s'CCS' = the Unisys CCS number used with CCS_TO_CCS transliteration

s'ISO' = the ISO CCS number used for interoperability

s'SDF' = the system data format (SDF) number used as an SDF character set type (code type) in SDF control records and data records

ccs_number

is the number of the CCS associated with the locale. If the service routine returns an error status other than OK, this value is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

LOCALE_NAME_TO_CCS_NUM can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 26 – S'LOCALE_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 45 – S'CCS_NUMBER_FORMAT_UNDEFINED'

Sample Code

This code example shows how to use LOCALE_NAME_TO_CCS_NUM to return the CCS number associated with the locale name. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure
" LOCALE_NAME_TO_CCS_NUM, isl_error_def, and
" isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for LOCALE_NAME_TO_CCS_NUM call:
" return status, locale name, and returned CCS.
"
DECLARE ret_stat        : isl_error_def;
DECLARE locale_name : 12 ASCII CHARACTERS LOCATABLE;
DECLARE ccs_number  : SIGNED WORD INTEGER LOCATABLE;
"
  locale_name := 'fr_FR.8859-1';
"
" The following parameters are passed:
"     Pointer to the string
"     Character position of first character in locale name
"     Number of characters in locale name
"     CCS number and format
"
  ret_stat := LOCALE_NAME_TO_CCS_NUM(LOC(locale_name),1,12,
                                     s'SDF',%ccs_number);
"
  IF ret_stat.isl_error_status = S'OK'
    THEN
       .
       . normal return
       .
    ELSE
      BEGIN
      IF ret_stat.isl_severity EQ S'isl_warning'
        THEN
          PUT FILE ('SYSPRINT') LIST ('Warning ',
                     ret_stat.isl_error_num, ' Received');
        ELSE
          IF ret_stat.isl_severity NE S'isl_successful'
            THEN
              PUT FILE ('SYSPRINT') LIST ('Error ',
                         ret_stat.isl_error_num, ' Received');
      END;
       .
       .
```

```
.
TERM
```

7.3.20. LOCALE_NAME_TO_LOCALE_NUM Service Routine

Entry Points

For Basic Mode PLUS: PLS$LNM2LNB

For Extended Mode PLUS: UPLS$LNM2LNB

Description

LOCALE_NAME_TO_LOCALE_NUM transforms a locale name to its corresponding locale number.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$lnm2lnb'` to `'upls$lnm2lnb'`.

```
PROCEDURE LOCALE_NAME_TO_LOCALE_NUM(
                                    string_ptr : WORD MACHINE POINTER,
                                    string_start_char : WORD SIGNED INTEGER,
                                    string_num_of_char : WORD SIGNED INTEGER,
                                   %locale_number : WORD SIGNED INTEGER)
      IMPORTED('pls$lnm2lnb')
      RESULT  isl_error_def;
```

Parameters

string_ptr

is a pointer to the string containing the locale name. This name is required by the SETLOCALE service routine. It is case-sensitive and must correspond to (1) a LOCALE_NAME entity in the Repository for ClearPath OS 2200 and (2) a locale that is active on the system.

string_start_char

is the character position in the string where the first character of the name is located. The position of the first character in the string is 1.

string_num_of_char

is the number of characters in the name.

locale_number

is the locale number corresponding to the specified name. If the service routine returns an error status other than OK, the value for locale_number is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

LOCALE_NAME_TO_LOCALE_NUM can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 26 – S'LOCALE_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'

Sample Code

This code example shows how to use LOCALE_NAME_TO_LOCALE_NUM to transform a locale name to its corresponding locale number. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure
" LOCALE_NAME_TO_LOCALE_NUM, isl_error_def, and
" isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for the LOCALE_NAME_TO_LOCALE_NUM call:
" the return status, the string pointer, and the character
" length of the name.
"
DECLARE ret_stat            : isl_error_def;
DECLARE locale_name     : 12 ASCII CHARACTERS LOCATABLE;
DECLARE locale_number : SIGNED WORD INTEGER LOCATABLE;
"
  locale_name := 'fr_FR.8859-1';
"
" The following parameters are passed:
"     Pointer to the string
"     Character position of first character in locale name
"     Number of characters in locale name
"     Locale number
"
  ret_stat := LOCALE_NAME_TO_LOCALE_NUM(LOC(locale_name),1,12,
              %locale_number);
  IF ret_stat.isl_error_status = S'OK'
    THEN
```

```
            .
            . process the value
            .
          ELSE
            BEGIN
            IF ret_stat.isl_severity EQ S'isl_warning'
              THEN
                PUT FILE ('SYSPRINT') LIST ('Warning ',
                           ret_stat.isl_error_num, ' Received');
              ELSE
                IF ret_stat.isl_severity NE S'isl_successful'
                  THEN
                    PUT FILE ('SYSPRINT') LIST ('Error ',
                               ret_stat.isl_error_num, ' Received');
            END;
              .
              .
              .
      TERM
```

7.3.21. LOCALE_NUM_TO_CCS_NUM Service Routine

Entry Points

For Basic Mode PLUS: PLS$LNB2CNB

For Extended Mode PLUS: UPLS$LNB2CNB

Description

LOCALE_NUM_TO_CCS_NUM returns the coded character set (CCS) number associated with the locale number.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$lnb2cnb'` to `'upls$lnb2cnb'`.

```
DEFINE TYPE isl_ccs_number_format = WORD STATUS
                                   (S'CCS' : 0,
                                    S'ISO' : 1,
                                    S'SDF' : 2);
PROCEDURE LOCALE_NUM_TO_CCS_NUM(
                                  locale_number : WORD SIGNED INTEGER,
                                  ccs_number_format : isl_ccs_number_format,
                                 %ccs_number : WORD SIGNED INTEGER)
          IMPORTED('pls$lnb2cnb')
          RESULT  isl_error_def;
```

Parameters

locale_number

is the number of a locale that is active on the system.

ccs_number_format

is the kind of CCS number desired:

s'CCS' = the Unisys CCS number for use with CCS_TO_CCS transliteration

s'ISO' = the ISO CCS number for interoperability use

s'SDF' = the system data format (SDF) number for use as an SDF character set type (code type) in SDF control records and data records

ccs_number

is the number of the CCS associated with the locale. If the service routine returns an error status other than OK, the value for ccs_number is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

LOCALE_NUM_TO_CCS_NUM can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 26 – S'LOCALE_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'NVALID_EM_ENVIRONMENT'
- 45 – S'CCS_NUMBER_FORMAT_UNDEFINED'

Sample Code

This code example shows how to use LOCALE_NUM_TO_CCS_NUM to return the CCS number associated with the locale number. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure
" LOCALE_NUM_TO_CCS_NUM, isl_error_def, and
" isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
```

```
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for LOCALE_NUM_TO_CCS_NUM call:
" return status, locale number, and CCS number.
"
DECLARE ret_stat            : isl_error_def;
DECLARE locale_number : SIGNED WORD INTEGER LOCATABLE;
DECLARE ccs_number      : SIGNED WORD INTEGER LOCATABLE;
"
  locale_number := 4;
"
" The following parameters are passed:
"     Locale number
"     CCS number
"
  ret_stat := LOCALE_NUM_TO_CCS_NUM(locale_number,s'SDF',%ccs_number);
"
  IF ret_stat.isl_error_status = S'OK'
    THEN
       .
       . normal return
       .
    ELSE
      BEGIN
      IF ret_stat.isl_severity EQ S'isl_warning'
        THEN
          PUT FILE ('SYSPRINT') LIST ('Warning ',
                     ret_stat.isl_error_num, ' Received');
        ELSE
          IF ret_stat.isl_severity NE S'isl_successful'
            THEN
              PUT FILE ('SYSPRINT') LIST ('Error ',
                         ret_stat.isl_error_num, ' Received');
      END;
       .
       .
       .
TERM
```

7.3.22. LOCALE_NUM_TO_LOCALE_NAME Service Routine

Entry Points

For Basic Mode PLUS: PLS$LNB2LNM

For Extended Mode PLUS: UPLS$LNB2LNM

Description

LOCALE_NUM_TO_LOCALE_NAME transforms a locale number to its corresponding locale name.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$lnb2lnm'` to `'upls$lnb2lnm'`.

```
PROCEDURE LOCALE_NUM_TO_LOCALE_NAME(
                                locale_number : WORD SIGNED INTEGER,
                                string_ptr : WORD MACHINE POINTER,
                                string_start_char : WORD SIGNED INTEGER,
                               %string_num_of_char : WORD SIGNED INTEGER)
        IMPORTED('pls$lnb2lnm')
        RESULT  isl_error_def;
```

Parameters

locale_number

is the number of a locale that is active on the system.

string_ptr

is a pointer to the string into which the service routine should write the corresponding locale name. This name is required for the SETLOCALE service routine. It is case-sensitive and corresponds to a LOCALE_NAME entity in the Repository for ClearPath OS 2200. If the service routine returns an error status other than OK, this value is undefined.

string_start_char

is the character position in the string where the first character of the name should be written. The position of the first character in the string is 1.

string_num_of_char

Upon entry to the service routine, this integer should be set to the character length of the buffer pointed to by string_ptr.

If the service routine succeeds, it sets string_num_of_char to the number of characters in the locale name.

If the service routine fails, this value is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

LOCALE_NUM_TO_LOCALE_NAME can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 26 – S'LOCALE_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'NVALID_EM_ENVIRONMENT'

Sample Code

This code example shows how to use LOCALE_NUM_TO_LOCALE_NAME to transform a locale number to its corresponding locale name. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure
" LOCALE_NUM_TO_LOCALE_NAME, isl_error_def, and
" isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for LOCALE_NUM_TO_LOCALE_NAME call:
" return status, string pointer, and character length of output
" buffer.
"
DECLARE ret_stat              : isl_error_def;
DECLARE locale_name       : 16 ASCII CHARACTERS LOCATABLE;
DECLARE locale_number    : SIGNED WORD INTEGER LOCATABLE;
DECLARE locale_name_len : SIGNED WORD INTEGER LOCATABLE;
"
  locale_number := 6;
  locale_name_len := 16;
"
" The following parameters are passed:
"     Locale number
"     Pointer to locale name
"     Character position of first character in locale name
"     Number of characters in locale name
"
  ret_stat := LOCALE_NUM_TO_LOCALE_NAME(locale_number,LOC(locale_name),
              1,%locale_name_len);
"
  IF ret_stat.isl_error_status = S'OK'
    THEN
       .
       .
       .
    ELSE
      BEGIN
      IF ret_stat.isl_severity EQ S'isl_warning'
```

```
          THEN
            PUT FILE ('SYSPRINT') LIST ('Warning ',
                       ret_stat.isl_error_num, ' Received');
          ELSE
            IF ret_stat.isl_severity NE S'isl_successful'
              THEN
                PUT FILE ('SYSPRINT') LIST ('Error ',
                           ret_stat.isl_error_num, ' Received');
        END;
         .
         .
         .
TERM
```

7.3.23. NAME_AND_NUMBER Service Routine

Entry Points

For Basic Mode PLUS: PLS$NMNB

For Extended Mode PLUS: UPLS$NMNB

Description

The NAME_AND_NUMBER packet interface provides a general interface for obtaining name-and-number information for various locales and CCSs. The Virtual Machine for the Java (TM) Platform on ClearPath OS 2200 can use this interface to obtain this information, which also includes the number of bits in a character for the requested locale or CCS.

Note: *All locales have an associated CCS number. However, not all CCSs are associated with a locale.*

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$nmnb'` to `'upls$nmnb'`.

```
PROCEDURE ISL_NMNB(%isl_error_def, %isl_nmnb_information_packet)
          IMPORTED('pls$nmnb');
```

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

NAME_AND_NUMBER can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 2 – S'INVALID_PACKET_VERSION'
- 15 – S'TOO_MANY_PARAMETERS'
- 16 – S'TOO_FEW_PARAMETERS'

- 26 – S'LOCALE_NOT_FOUND'
- 31 – S'CCS_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 40 – S'CCS_NUMBER_NOT_UNIQUE'
- 45 – S'CCS_NUMBER_FORMAT_UNDEFINED'
- 57 – S'INVALID_NMNB_IDENTIFIER'
- 58 – S'INVALID_NMNB_NAME_LENGTH'

Information Packet Structure

See 7.2.3 for the description of the NAME_NUMBER information packet.

See the NAME_AND_NUMBER Packet Interface in the COBOL section for the list of fields in the *name_and_number* information packet.

The following code defines the structure of the NAME_AND_NUMBER information packet for internationalized OS 2200 applications that are written in Basic or Extended Mode PLUS.

```
DEFINE TYPE isl_nmnb_information_packet = MAPPED LOCATABLE
            [ isl_nmnb_reserved_1           : 18 BIT UNSIGNED INTEGER,
              isl_nmnb_identifier           : 6 BIT UNSIGNED INTEGER,
              isl_nmnb_char_bit_size        : 6 BIT UNSIGNED INTEGER,
              isl_nmnb_version              : 6 BIT UNSIGNED INTEGER,
              isl_nmnb_number               : WORD SIGNED INTEGER,
              isl_nmnb_ccs_name_length      : WORD SIGNED INTEGER,
              isl_nmnb_ccs_number           : WORD SIGNED INTEGER,
              isl_nmnb_ccs_iso_number       : WORD SIGNED INTEGER,
              isl_nmnb_ccs_sdf_number       : WORD SIGNED INTEGER,
              isl_nmnb_ccs_name             : 16 ASCII CHARACTERS,
              isl_nmnb_locale_name_length   : WORD SIGNED INTEGER,
              isl_nmnb_locale_number        : WORD SIGNED INTEGER,
              isl_nmnb_locale_name          : 16 ASCII CHARACTERS ];
DEFINE isl_nmnb_information_packet_length = 16;
" ISL_NMNB_IDENTIFIER field values
DEFINE isl_ccs_nb_id = 1;
DEFINE isl_ccs_iso_nb_id = 2;
DEFINE isl_ccs_sdf_nb_id = 3;
DEFINE isl_locale_nb_id = 4;
DEFINE isl_ccs_nm_id = 5;
DEFINE isl_locale_nm_id = 6;
```

Sample Code

This code example shows how to use NAME_AND_NUMBER to obtain CCS and locale name-and-number information. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
module;
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
DECLARE nmnb_pkt : isl_nmnb_information_packet;
DECLARE stat : isl_error_def;
"
open file 'sysprint' stream output;
put file 'sysprint' list('start of test - name & number packet
interface');
"
put file 'sysprint' list('*** test of CCS number input ***');
nmnb_pkt.isl_nmnb_reserved_1 := 0;
nmnb_pkt.isl_nmnb_version := 1;
nmnb_pkt.isl_nmnb_identifier := 1;
nmnb_pkt.isl_nmnb_number := 35;
ISL_NMNB(%stat, %nmnb_pkt);
IF stat.isl_error_status EQ S'OK' THEN
  BEGIN
    put file 'sysprint' list('char bit size
',nmnb_pkt.isl_nmnb_char_bit_size);
    put file 'sysprint' list('ccs iso number
',nmnb_pkt.isl_nmnb_ccs_iso_number);
    put file 'sysprint' list('ccs sdf number
',nmnb_pkt.isl_nmnb_ccs_sdf_number);
    put file 'sysprint' list('ccs name length
',nmnb_pkt.isl_nmnb_ccs_name_length);
    put file 'sysprint' list('ccs name ',nmnb_pkt.isl_nmnb_ccs_name);
  END;
ELSE
  BEGIN
    put file 'sysprint' list('isl__nmnb errored');
    put file 'sysprint' list('status is ', stat.isl_error_num);
  END;
"
"
put file 'sysprint' list('test complete');
"
term
```

7.3.24. NL_LANGINFO Service Routine

Entry Points

For Basic Mode PLUS: PLS$NLLANG

For Extended Mode PLUS: UPLS$NLLANG

Description

NL_LANGINFO returns a string containing relevant information about the particular language or cultural area defined in the program's locale.

NL_LANGINFO requires a specific locale item number so it can determine what locale information to return.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$nllang' to 'upls$nllang'`.

```
PROCEDURE nl_langinfo (item            : nl_langinfo_item,
                       str_out_ptr     : WORD MACHINE POINTER,
                       str_out_start   : WORD SIGNED INTEGER,
                       %str_out_length : WORD SIGNED INTEGER
                                            LOCATABLE)
          IMPORTED('pls$nllang')
          RESULT isl_error_def;
```

Parameters

item

is a status variable indicating which locale item to retrieve. In a locale where the particular locale data is not defined, the service routine returns a string to the corresponding string in the POSIX locale.

str_out_ptr

is a pointer to the output buffer where the resulting string is written.

str_out_start

is the starting character position for the output buffer . The first position is 1.

str_out_length

is set before the call to the service routine to indicate the output buffer length in characters. On normal completion, this variable contains the number of characters in the resulting string.

RESULT

is the error structure returned by the service routine(see 7.2).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

NL_LANGINFO can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 20 – S'EV_NAME_INVALID_CHAR'

- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'
- 50 – S'INVALID_LOCALE_ITEM'

Sample Code

This code example shows how to use NL_LANGINFO to return a string containing information about a particular language or cultural area defined in the program's locale. The call returns the full name of the eighth month in the year, based on the POSIX locale, which specifies the minimal environment for C-language translation.

For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declarations for nl_langinfo,
" nl_langinfo_item, isl_error_def, and isl_error_def.
"
COPY('ISL-ERR-COPY/PLS', NO SOURCE);
COPY('ISL-ROU-COPY/PLS', NO SOURCE);
"
" Declare variables for the nl_langinfo call.
"
DECLARE str_out         : 15 CHARACTERS LOCATABLE;
DECLARE str_out_length  : WORD SIGNED INTEGER LOCATABLE;
DECLARE ret_stat        : isl_error_def;
BEGIN
   OPEN FILE ('SYSPRINT') STREAM OUTPUT;
   "
   " Initialize parameters.
   "
   str_out := ' ';
   str_out_length := BYTESIZE(str_out);
   "
   " Perform call to nl_langinfo.
   "
   ret_stat := nl_langinfo(S'ISL_MON_8',LOC(str_out),1,
      %str_out_length);
   "
```

```
   " Display result.
   "
   IF ret_stat.isl_error_status = S'OK' THEN
      PUT FILE ('SYSPRINT') LIST (str_out);
END;
TERM
```

Assuming that the locale for LC_TIME is "POSIX", the following output is displayed:

```
August
```

7.3.25. OPEN_CCS_TO_CCS Service Routine

Entry Points

For Basic Mode PLUS: PLS$OPENCCS

For Extended Mode PLUS: UPLS$OPENCCS

Description

OPEN_CCS_TO_CCS, CCS_TO_CCS, and CLOSE_CCS_TO_CCS are the three service routines used to transliterate a character string from one coded character set (CCS) to another. The OPEN_CCS_TO_CCS service routine define the conversion descriptor used by the CCS_TO_CCS service routine to identify the transliteration to be performed.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$openccs'` to `'upls$openccs'`.

```
PROCEDURE open_ccs_to_ccs(
                          from_ccs : WORD SIGNED INTEGER,
                          to_ccs : WORD SIGNED INTEGER,
                          %error_status : isl_error_def)
          IMPORTED('PLS$OPENCCS')
          RESULT  WORD SIGNED INTEGER;
```

Parameters

from_ccs

is an integer that identifies the CCS to be transliterated.

to_ccs

is an integer that identifies the resulting CCS.

error_status

is the error status returned by the service routine.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

RESULT

is the conversion descriptor (internal identifier for the transliteration tables) returned by OPEN_CCS_TO_CCS.

Error Statuses

See 7.3.3 for the error statuses.

Sample Code

See 7.3.3 for sample code.

7.3.26. PUTENV Service Routine

Entry Points

For Basic Mode PLUS: PLS$PUTENV

For Extended Mode PLUS: UPLS$PUTENV

Description

PUTENV sets the current value of the environment variable for the run level.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$isalnum'` to `'upls$isalnum'`.

```
PROCEDURE putenv(
                env_var_name_ptr : WORD MACHINE POINTER,
                env_var_name_length : WORD SIGNED INTEGER,
                env_var_value_ptr : WORD MACHINE POINTER,
                env_var_value_length : WORD SIGNED INTEGER)
          IMPORTED('pls$putenv')
          RESULT  isl_error_def;
```

Parameters

env_var_name_ptr

is a pointer to a string containing the environment variable name.

env_var_name_length

is the character length of the string pointed to by env_var_name_ptr.

env_var_value_ptr

is a pointer to a string containing the value setting for the environment variable.

env_var_value_length

is the character length of the string pointed to by env_var_value_ptr.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

PUTENV can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 22 – S'EV_AREA_FULL'
- 23 – S'EV_NAME_ILLEGAL_FOR_TIP'
- 25 – S'EV_VALUE_TOO_LONG'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 38 – S'PUT_ENV_ERROR

Sample Code Declaration (Basic and Extended Mode PLUS)

Note: *Include this declaration with the* PUTENV examples.

This code example shows how to use PUTENV to set environment variables (EV) at the run level. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
"
" Copy elements containing the declaration of procedure
" PUTENV, isl_error_def, and isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" PUTENV call:  Declare variables for return status and
" EV string.
"
DECLARE ret_stat                        : isl_error_def;
DECLARE env_var_name_string  : 6 CHARACTERS LOCATABLE;
DECLARE env_var_value_string  : 12 CHARACTERS LOCATABLE;
```

Example 1: Setting EV Name and Value (Basic and Extended Mode PLUS)

Note: *Precede this declaration with the PUTENV declaration.*

```
"
" LC_ALL is the EV name; fr_FR.8859-1 is the EV value.
"
env_var_name_string := 'LC_ALL';
env_var_value_string := 'fr_FR.8859-1';
"
" The following parameters are passed:
"     Pointer to EV name
"     EV name length=6
"     Pointer to EV value
"     EV value length=12
"
ret_stat := putenv(LOC(env_var_name_string),
                   BYTESIZE(env_var_name_string),
                   LOC(env_var_value_string),
                   BYTESIZE(env_var_value_string));
"
" Check return status; it should be OK.
"
IF ret_stat.isl_error_status NE S'OK'
  THEN
    IF ret_stat.isl_severity EQ S'ISL_WARNING'
      THEN
        PUT FILE ('SYSPRINT') LIST ('Warning ',
                   ret_stat.isl_error_num, ' received');
      ELSE
        PUT FILE ('SYSPRINT') LIST ('ERROR ',
                   ret_stat.isl_error_num, ' received');
```

Code Example 2: Error in Setting EV Name and Value (Basic and Extended Mode PLUS)

Note: *Precede this routine with the PUTENV declaration.*

```
" LC_ALL is EV name; fr_FR.8859-1 is the EV value.
" A blank in the EV name creates an error condition.
         .
         .
         .
env_var_name_string := 'LC ALL';
env_var_value_string := 'fr_FR.8859-1';
"
" The following parameters are passed:
"     Pointer to EV name
"     EV name length=8
"     Pointer to EV value
"     EV value length=12
"
ret_stat :=
```

```
putenv(LOC(env_var_name_string),6,LOC(env_var_value_string),12);
"
" Check return status; error 20 should be displayed.
"
IF ret_stat.isl_error_status NE S'OK'
  THEN
    IF ret_stat.isl_severity EQ S'ISL_WARNING'
      THEN
        PUT FILE ('SYSPRINT') LIST ('Warning ',
                   ret_stat.isl_error_num, ' received');
      ELSE
        PUT FILE ('SYSPRINT') LIST ('ERROR ',
                   ret_stat.isl_error_num, ' received');
```

7.3.27. SETLOCALE Service Routine

Entry Points

For Basic Mode PLUS: PLS$SETLOC

For Extended Mode PLUS: UPLS$SETLOC

Description

SETLOCALE is used to set category values and query locale information for a user program.

Once called by the program, SETLOCALE establishes the locale for the current activity. This operation includes such functions as specifying the locale name and the length and storage location for this name. See Section 1 for information about locale names and Section 9 for the list of valid locale names

The I18NLIB routines rely on compile-time parameter checking. I18NLIB provides a copy element to define the data structures, types, and procedures. For Basic Mode PLUS, the copy element is *isl-rou-copy/pls* and for Extended Mode PLUS the copy element is *isl-rou-copy/upls*. The applicable parts of the copy element for this service routine are listed below. For Extended Mode PLUS, the procedure entry point is *upls$setloc* instead of *pls$setloc*.

```
DEFINE TYPE category_status_def = WORD STATUS
      (S'LC_ALL'      : 0,
       S'LC_COLLATE'  : 1,
       S'LC_CTYPE'    : 2,
       S'LC_MESSAGES' : 3,
       S'LC_MONETARY' : 4,
       S'LC_NUMERIC'  : 5,
       S'LC_TIME'     : 6);
PROCEDURE setlocale(category : category_status_def,
                    requested_locale_ptr : WORD MACHINE POINTER,
                    requested_locale_length : WORD SIGNED INTEGER,
                    resulting_locale_ptr : WORD MACHINE POINTER,
                    %resulting_locale_length : WORD SIGNED INTEGER)
```

```
IMPORTED('pls$setloc')
RESULT  isl_error_def;
```

Parameters

category

is a status that specifies the portion of a program's locale to be set or queried.

There are seven allowable categories: S'LC_ALL, S'LC_COLLATE, S'LC_CTYPE, S'LC_MESSAGES, S'LC_MONETARY, S'LC_NUMERIC, and S'LC_TIME.

requested_locale_ptr

is a pointer to a string that specifies a locale name, "_DEFAULT", or "_QUERY".

The maximum length for a locale name is 16 characters. When dealing with locale names, SETLOCALE is case-sensitive and ignores trailing spaces.

requested_locale_length

is the length, in characters, of the requested locale name.

resulting_locale_ptr

is a pointer to a string that specifies the name of the locale that is set or queried for by requested_locale_ptr. If this operation is not successful, the pointer is undefined. This pointer may be the same as the one passed as the requested_locale_ptr.

resulting_locale_length

is the length, in characters, of the resulting locale name. Upon entry to SETLOCALE, this parameter is equal to the length, in characters, of the buffer indicated by the resulting_locale_ptr parameter passed to hold the resulting locale name. If the SETLOCALE operation is not successful, this length is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A or a detailed description of the error statuses returned.

SETLOCALE can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'

- 28 – S'INVALID_CATEGORY'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use SETLOCALE to establish a locale for an activity. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing declaration of procedure SETLOCALE.
"
COPY('isl-err-copy/pls',NO SOURCE);
COPY('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the SETLOCALE calls.
"
DECLARE locale : 20 CHARACTERS LOCATABLE;
DECLARE length : WORD SIGNED INTEGER LOCATABLE;
DECLARE status : isl_error_def;
"
" Set all the categories to their default values.
"
  locale := '_DEFAULT';
  length := 20;
  status := setlocale(S'LC_ALL',LOC(locale),8,LOC(locale),%length);
"
" Query to determine the locale for category LC_COLLATE.
"
  locale := '_QUERY';
  length := 20;
  status := setlocale(S'LC_COLLATE',LOC(locale),6,LOC(locale),
            %length);
"
" If LC_COLLATE's locale is not Norwegian, set it to Norwegian.
"
  IF locale(1 TO length) NE 'no_NO.8859-1'
    THEN
      BEGIN
      locale := 'no_NO.8859-1';
```

```
        length := 20;
        status := setlocale(S'LC_COLLATE',LOC(locale),12,LOC(locale),
           %length);
        END;
       .
       .
       .
TERM
```

7.3.28. STRFMON Service Routine

Entry Points

For Basic Mode PLUS: PLS$STRFMON

For Extended Mode PLUS: UPLS$STRFMON

Description

STRFMON is used to convert numeric monetary values to a string representation based on a specified format and the locale information defined by the LC_MONETARY category.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$strfmon'` to `'upls$strfmon'`.

```
PROCEDURE strfmon (fmt_str_ptr : WORD MACHINE POINTER,
                   fmt_str_start : WORD SIGNED INTEGER,
                   fmt_str_length : WORD SIGNED INTEGER,
                   arguments_ptr : WORD SIGNED POINTER,
                   str_out_ptr : WORD MACHINE POINTER,
                  %str_out_length : WORD SIGNED INTEGER LOCATABLE)
          IMPORTED('pls$strfmon')
          RESULT  isl_error_def;
```

Parameters

fmt_str_ptr

is a pointer to the format string. This string has two objects: plain characters and conversion specifications.

Plain Characters: Plain characters are copied unchanged to the output buffer.

Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

fmt_str_start

is the starting character position in the format string. The first position is 1.

fmt_str_length

is set to the length in characters of the format string.

arguments_ptr

is an array of one or more consecutive LONG WORD SIGNED REAL values that are converted to a monetary formatted string using the format string. For details, go to the Conversion Specification Table.

str_out_ptr

is a pointer to the output buffer where the resulting string is written.

str_out_start

is the starting characters in the output buffer where the resulting string is written. The first position is 1.

str_out_length

is set before the call to the service routine to indicate the output buffer length in characters. If the total number of resulting characters is not more than indicated by this variable, it will contain the number of characters in the resulting string. This number matches the number of characters placed in the output buffer.

RESULT

is the error structure returned by the service routine (see 7.2.1).

Conversion Specification Table (STRFMON)

The following table shows the sequence of arguments in a conversion specification. Each conversion specification results in the fetching of zero or more sequential arguments that are converted and formatted.

- Each argument must be a double-precision floating point number.
- If excess arguments remain after the format string is exhausted, they are ignored.
- If the format string contains more conversion specifications than arguments, the results are unpredictable. With the C compiler, an error is returned.

Note: *The % character and the conversion character are required; all others are optional***.**

Table 7–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
1	% character	Required argument
2	flags	Optional arguments
	=f	A numeric fill character. This character (1) must be representable in a single byte; (2) does not affect field width filling, which always uses the space character; and (3) is ignored unless left precision is specified. The default is the space character.
	^	Indicates that the currency amount should not be formatted with currency characters. If this flag is defined for the current locale, grouping characters are inserted.
	+ or (	Specifies the style for representing positive and negative currency amounts. Only one style can be specified. If +, the locale's equivalent of + and - are used. If (, negative amounts are enclosed in parentheses ( ). If neither style is specified, the default is +.
	!	Suppresses the currency symbol from output conversion.
	=	Specifies the alignment. If it is present, all fields are left justified.
3	field width (w)	Optional argument. A decimal digit that specifies a minimum field width in bytes. If the - flag is specified, the result is left-justified. If the - flag is not specified, the result is right-justified. The default is zero (0).
4	left precision (#n)	Optional argument. A pound sign followed by a decimal digit string that specifies the maximum number of digits expected to be formatted to the left of the radix character. It can be used to (1) keep formatted output from multiple calls to STRFMON aligned in the same columns and (2) fill unused positions with a special character. • If more than n digit positions are required, this value is ignored, and the excess positions are filled with numeric fill characters (=f). • If the grouping has not been suppressed with the ^ flag and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit. • To ensure alignment, any characters appearing before or after the number in the formatted output (such as currency or sign symbols) are padded as necessary with space characters to make their positive and negative formats an equal length.

Table 7–2. Sequence of Arguments in a Conversion Specification

Order	Argument	Description
5	right precision (.p)	Optional argument. A period followed by a decimal digit that specifies the number of digits after the radix character. If the value is zero, no radix character appears . The amount being formatted is rounded to the specified number of digits before formatting occurs. If right precision is not included, a default specified by the current locale is used.
6	conversion character	Required argument
	i	This argument should use the locale's international currency format. USA example: `USD 1,234.56`
	n	This argument should use the locale's national currency format. USA example: `$1,234.56`
	%	This means no argument is converted. The entire conversion specification must be %%.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

STRFMON can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'
- 51 – S'INVALID_CONVERSION_SPEC'
- 52 – S'INVALID_ARGUMENT_VALUE'

Sample Code

This code example shows how to use STRFMON to format a monetary value based on the POSIX locale, which specifies the minimal environment for C-language translation. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declarations for STRFMON and
" isl_error_def.
"
COPY('isl-err-copy/pls',NO SOURCE);
COPY('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the STRFMON call.
"
DECLARE fmt_str         : 35 CHARACTERS LOCATABLE;
DECLARE str_out         : 40 CHARACTERS LOCATABLE;
DECLARE str_out_length  : WORD SIGNED INTEGER LOCATABLE;
DECLARE ret_stat        : isl_error_def;
"
" Declare arguments for the strfmon call.  They must be
" consecutive and declared using LONG WORD SIGNED REAL.
" Note that the first in the list should be declared as
" LOCATABLE as well.
"
DECLARE arguments(2)    : LONG WORD SIGNED REAL LOCATABLE;
"
BEGIN
   OPEN FILE ('sysprint') STREAM OUTPUT;
"
" Initialize parameters.
"
   fmt_str := 'Cost is %=*#5n (%#5.0n rounded).';
   str_out := ' ';
   str_out_length := BYTESIZE(str_out);
   arguments(1) := 123.45;
   arguments(2) := 123.45;
"
" Perform call to strfmon.
"
   ret_stat := strfmon(LOC(fmt_str),1,35,LOC(arguments(1)),
                       LOC(str_out),1,%str_out_length);
"
" Display result.
"
   IF ret_stat.isl_error_status = S'OK' THEN
      PUT FILE ('sysprint') LIST (str_out);
END;
TERM
```

Assuming that the locale for LC_MONETARY is "POSIX", the following output is displayed:

```
Cost is $***123.45 ($  123 rounded).
```

7.3.29. STRFTIME Service Routine

Entry Points

For Basic Mode PLUS: PLS$STRFTIM

For Extended Mode PLUS: UPLS$STRFTIM

Description

STRFTIME is used to convert date and time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$strftim'` to `'upls$strftim'`.

```
PROCEDURE strftime (fmt_str_ptr     : WORD MACHINE POINTER,
                    fmt_str_start   : WORD SIGNED INTEGER,
                    fmt_str_length  : WORD SIGNED INTEGER,
                    date_time_ptr   : WORD MACHINE POINTER,
                    str_out_ptr     : WORD MACHINE POINTER,
                    str_out_start   : WORD SIGNED INTEGER,
                   %str_out_length : WORD SIGNED INTEGER LOCATABLE)
          IMPORTED('pls$strftim')
          RESULT  isl_error_def;
```

Parameters

fmt_str_ptr

is a pointer to the format string. This string has two objects: plain characters and conversion specifications.

Plain Characters: Plain characters are copied unchanged to the output buffer.

Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. See Table 7–2 for a description of the conversion specifications.

fmt_str_start

is the starting character position in the format string. The first position is 1.

fmt_str_length

is the number of characters in the format string.

date_time_ptr

is a pointer to the date/time structure packet containing the integer values to be converted. See **Error! Reference source not found.** for a description of this structure

str_out_ptr

is a pointer to the output buffer where the converted date and time characters are to be stored.

str_out_start

is the starting character position in the output buffer where the formatted string is to begin. The first position is 1.

str_out_length

Upon entering the service routine, this integer should be set to the maximum length of the output buffer.

- If the service routine succeeds, it sets str_out_length to the byte length of the formatted string.
- If the service routine fails, the value is undefined.

RESULT

is the error structure returned by the service routine (see 7.2.1).

Conversion Specification Table (STRFTIME)

Each conversion specification (CS) is replaced by the appropriate characters listed in this table. These characters are determined by the current locale and by the values contained in the date-and-time structure. If a CS does not correspond to any of the ones listed in this table, the behavior is undefined.

Table 7–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%a	Weekday name, abbreviated	abday	tm_wday
%A	Weekday name, full	day	tm_wday
%b	Month name, abbreviated	abmon	tm_mon
%B	Month name, full	mon	tm_mon
%c	Appropriate date/time representation	d_t_fmt	Varies by locale

Table 7–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%C	Century number. To get this number, the year is divided by 100 and truncated to an integer as a decimal [00-99].	n/a	tm_year
%d	Day of month as decimal [01,31]	n/a	tm_mday
%D	Equivalent to %m/%d/%y	n/a	tm_mon, tm_mday, and tm_year.
%e	Day of month as decimal [1,31]. A single digit is preceded by a space.	n/a	tm_mday
%F	Equivalent to %Y-%m-%d (the ISO8601:2000 standard date format)	n/a	tm_year, tm_mon, and tm_mday
%g	Replaced by the last 2 digits of the week-based year as a decimal number [00-99].	n/a	tm_year, tm_yday, and tm_mday
%G	Replaced by the week-based year and as a decimal number.	n/a	tm_year, tm_yday, tm_mday
%h	Month name, abbreviated	abmon	tm_mon
%H	Hour on 24-hour clock as decimal [00,23]	n/a	tm_hour
%I	Hour on 12-hour clock as decimal [01,12]	n/a	tm_hour
%j	Day of year as decimal [001,366]	n/a	tm_yday
%m	Month as decimal [01,12]	n/a	tm_mon
%M	Minute as decimal [00,59]	n/a	tm_min
%n	New line character (012)	n/a	n/a
%p	Equivalent for ante-meridiem (AM) and post-meridiem (PM)	am_pm	tm_hour
%r	Time in AM and PM notation; equivalent of %I:%M:%S %p in the POSIX locale	t_fmt_ampm	Refer to individual CSs.
%R	Time in 24-hour notation (%H:%M)	n/a	tm_hour and tm_min.
%S	Second as decimal [00,60]	n/a	tm_sec
%t	Tab character	n/a	n/a
%T	Time (%H:%M:%S)	n/a	tm_hour, tm_min, and tm_sec.
%u	Weekday as decimal [1,7]	n/a	tm_wday
%U	Week number of the year as decimal [00,53] with Sunday as day 1	n/a	tm_yday and tm_wday

Table 7–3. STRFTIME Conversion Specification Table

This CS	Is replaced by these characters for the current locale	And references these LC_TIME keywords	And these date/time structure fields
%V	Week number of the year as decimal [01,53] with Monday as day 1 If the week containing January 1 has four or more days in the new year, then it is considered week 1. If is does not, then it is the last week of the previous year, and the next week is week 1.	n/a	tm_year, tm_mday, and tm_yday
%w	weekday as decimal [0,6] with 0 representing Sunday	n/a	tm_wday
%W	Week number of the year as decimal [00,53] with Monday as day 1. All days in a new year preceding the first Sunday are considered to be in week 0.	n/a	tm_wday and tm_yday
%x	Appropriate date representation	d_fmt	Varies by locale
%X	Appropriate time representation	t_fmt	Varies by locale
%y	Year without century as decimal [00,99]	n/a	tm_year
%Y	Year with century as decimal [1998]	n/a	tm_year
%z	Replaced by the offset from UTC in the ISO 8601:2000 standard format (+hhmm or -hhmm) or by no characters if no timezone is determinable.	n/a	tm_isdst
%Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists.	n/a	tm_isdst
%%	%	n/a	n/a

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

STRFTIME can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'

- 30 – S'CORRUPTED_TABLE'
- 34 – S'NVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'
- 51 – S'INVALID_CONVERSION_SPEC'

Sample Code

This code example shows how to use STRFTIME to format a date value to a string representation based on locale en_US.8859-1 (English as spoken in the US). For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure STRFTIME,
" and isl_date_time_def.
"
COPY ('isl-err-copy/pls', NO SOURCE);
COPY ('isl-rou-copy/pls', NO SOURCE);
"
" Declare the variables for the STRFTIME call; the return status,
" the string pointer, the buffer pointer, and the date_and_time
" structure.
"
DECLARE ret_stat     : isl_error_def;
DECLARE in_string    : 12 ASCII CHARACTERS LOCATABLE;
DECLARE out_buffer   : 32 ASCII CHARACTERS LOCATABLE;
DECLARE out_buf_len  : WORD SIGNED INTEGER;
DECLARE d_and_t      : isl_date_time_def;
"
in_string := 'Today is %x.';
d_and_t.tm_mon := 7;
d_and_t.tm_year := 101;      " The year 2001
d_and_t.tm_mday := 22;
out_buf_len := 32;
"
OPEN FILE 'SYSPRINT' STREAM OUTPUT;
ret_stat := STRFTIME(LOC(in_string),1,12,LOC(d_and_t),
                     LOC(out_buffer),1,%out_buf_len);
"
IF ret_stat.isl_error_status EQ S'OK'
   THEN
      PUT FILE ('SYSPRINT') LIST(out_buffer);
   ELSE
```

```
        IF ret_stat.isl_severity EQ S'ISL_WARNING'
           THEN
              PUT FILE ('SYSPRINT') LIST ('WARNING ',
                       ret_stat.isl_error_num, ' RECEIVED');
           ELSE
              IF ret_stat.isl_severity  NE S'ISL_SUCCESSFUL'
                 THEN
                    PUT FILE ('SYSPRINT') LIST ('ERROR ',
                             ret_stat.isl_error_num, 'RECEIVED');
                .
                .
                .
TERM
```

Assuming that the locale for LC_TIME is "en_US.8859-1", the following output is displayed:

```
Today is 08/22/01.
```

7.3.30. STRING_COLLATE Service Routine

Entry Points

For Basic Mode PLUS: PLS$STRCOLL

For Extended Mode PLUS: UPLS$STRCOLL

Description

STRING_COLLATE compares two character strings and returns their collating order. These strings can be up to 4096 characters long. The collating rules are determined by the current setting of the LC_COLLATE category.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$strcoll'` to `'upls$strcoll'`.

```
    DEFINE TYPE isl_collation_result = WORD STATUS
                                       (S'BEFORE' : 0,
                                        S'EQUAL' : 1,
                                        S'AFTER' : 2);
    PROCEDURE STRING_COLLATE(
                              string1_ptr : WORD MACHINE POINTER,
                              string1_start_char : WORD SIGNED INTEGER,
                              string1_num_of_char : WORD SIGNED INTEGER,
                              string2_ptr : WORD MACHINE POINTER,
                              string2_start_char : WORD SIGNED INTEGER,
                              string2_num_of_char : WORD SIGNED INTEGER,
                              %error_status : isl_error_def)
                 IMPORTED('pls$strcoll')
                 RESULT  isl_collation_result;
```

Parameters

string1_ptr

is a pointer to the first string containing characters to be compared.

string1_start_char

is the character position in the first string. This is the location of the first character to be included in the comparison. The position of the first character in the string is 1.

string1_num_of_char

is the number of characters in the first string to be included in the comparison. The maximum string length is 4096 characters.

string2_ptr

is a pointer to the second string containing the characters to be compared.

string2_start_char

is the character position in the second string. This is the location of the first character to be included in the comparison. The position of the first character in the string is 1.

string2_num_of_char

is the number of characters in the second string to be included in the comparison. The maximum string length is 4096 characters.

error_status

is the error structure returned by the service routine (see 7.2.1). If error_status is not OK, then RESULT is undefined.

isl_collation_result

is a status constant that indicates the result of the comparison. The possible values are

S'BEFORE' = First string collates before second string.

S'EQUAL' = Strings collate equally.

S'AFTER' = First string collates after second string.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

STRING_COLLATE can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 7 – S'ENCODING_NOT_SUPPPORTED'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 36 – S'STRING1_TOO_LONG'
- 37 – S'STRING2_TOO_LONG'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code example shows how to use STRING_COLLATE to compare two character strings and return their collating order. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure
" STRING_COLLATE, isl_error_def, and isl_error_status_def.
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare variables for the STRING_COLLATE call.  This includes
" the error status, the first string , the second string, and
" the collation result.
"
DECLARE ret_stat           : isl_error_def;
DECLARE in_string1       : 6 CHARACTERS LOCATABLE;
DECLARE in_string2       : 6 CHARACTERS LOCATABLE;
DECLARE collate_result  : isl_collation_result;
"
```

```
  in_string1:= 'resume';
  in_string2:= 'résumé';
"
" Two strings are compared.  The service routine determines
" the collation order by the rules determined by the setting
" of LC_COLLATE.  Different locales will cause the strings to
" collate differently.
"
" In this example, 'resume' will collate before 'résumé' if the
" ISO 8859.1 code set is used to set LC_COLLATE to the string
" 'fr_FR.8859-1', which corresponds to the French language as
" spoken in France.  If LC_COLLATE indicates some other locale,
" then the two " strings might collate in a different order.
"
  collate_result:= STRING_COLLATE(LOC(in_string1),1,6,LOC
        (in_string2),1,6,%ret_stat);
  IF ret_stat.isl_error_status = S'OK'
    THEN
      CASEENTRY collate_result
         CASE S'BEFORE':
           PUT FILE ('SYSPRINT') LIST('resume collates before
              résumé');
         CASE S'EQUAL':
           PUT FILE ('SYSPRINT') LIST('resume and résumé collate' &
              'equally');
         CASE S'AFTER':
           PUT FILE ('SYSPRINT') LIST('resume collates after
              résumé');
      ENDCASE;
    ELSE
      BEGIN
      IF ret_stat.isl_severity EQ S'isl_warning'
        THEN
          PUT FILE ('SYSPRINT') LIST('Warning ',
                     ret_stat.isl_error_num, ' Received');
        ELSE
          PUT FILE ('SYSPRINT') LIST('Error ',
                     ret_stat.isl_error_num, ' Received');
      END;
        .
        .
        .
TERM
```

7.3.31. STRING_TRANSFORM Service Routine

Entry Points

For Basic Mode PLUS: PLS$STRXFRM

For Extended Mode PLUS: UPLS$STRXFRM

Description

STRING_TRANSFORM converts a character string into an ordering key. This string can be up to 4096 characters long. The transformation rules are determined by the current setting of the LC_COLLATE category.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$strxfrm'` to `'upls$strxfrm'`.

```
      PROCEDURE string_transform(
                                string_ptr : WORD MACHINE POINTER,
                                string_start_char : WORD SIGNED INTEGER,
                                string_num_of_char : WORD SIGNED INTEGER,
                                out_buf_ptr : WORD MACHINE POINTER,
                                out_buf_start_byte : WORD SIGNED INTEGER,
                                %out_buf_num_of_bytes : WORD SIGNED
  INTEGER
                                                          : (LOCATABLE))
                IMPORTED('pls$strxfrm')
                RESULT  isl_error_def;
```

Parameters

string_ptr

is a pointer to the string containing the characters to be transformed.

string_start_char

is the character position in the string at which the first character to be transformed is located. The position of the first character in the string is 1.

string_num_of_char

is the number of characters to be transformed.

out_buf_ptr

is a pointer to the buffer in which STRING_TRANSFORM should write the transformed string. If the STRING_TRANSFORM operation fails, the contents of the buffer are undefined.

out_buf_start_byte

is the character position in the buffer at which the transformed string is to begin. The position of the first character in the string is 1.

out_buf_num_of_bytes

Upon entry to the service routine, this integer should be set to the maximum length of the buffer pointed to by out_buf_ptr.

If the service routine succeeds, it sets this value to the byte length of the transformed string.

If the service routine fails, this value is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

STRING_TRANSFORM can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 7 – S'ENCODING_NOT_SUPPORTED'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 36 – S'STRING1_TOO_LONG'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code routine shows how to use STRING_TRANSFORM to convert a character string to an ordering key. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure
" STRING_TRANSFORM, isl_error_def, and isl_error_status_def.
```

```
"
COPY('isl-err-copy/pls', NO SOURCE);
COPY('isl-rou-copy/pls', NO SOURCE);
"
" Declare two sets of variables for the STRING_TRANSFORM
" call.  Include the following in each set:  return status,
" string pointer, buffer pointer, and character length of
" output buffer.
"
DECLARE ret_stat1           : isl_error_def;
DECLARE in_string1         : 6 ASCII CHARACTERS LOCATABLE;
DECLARE out_buffer1       : 54 ASCII CHARACTERS LOCATABLE;
DECLARE out_buffer_len1 : SIGNED WORD INTEGER LOCATABLE;
"
DECLARE ret_stat2           : isl_error_def;
DECLARE in_string2         : 6 ASCII CHARACTERS LOCATABLE;
DECLARE out_buffer2       : 54 ASCII CHARACTERS LOCATABLE;
DECLARE out_buffer_len2 : SIGNED WORD INTEGER LOCATABLE;
"
  in_string1:= 'ABCDEF';
  out_buffer_len1 := 54;
"
" The following parameters are passed:
"     Pointer to string
"     Character position of first character to be transformed
"     Number of characters to be transformed
"     Pointer to output buffer
"     Character position of transformed string within buffer
"     Character length of output buffer
"
  ret_stat1:=STRING_TRANSFORM(LOC(in_string1),1,6,LOC(out_buffer1),
                              1,%out_buffer_len1);
"
  in_string2:= 'abcdef';
  out_buffer_len2 := 54;
"
" The following parameters are passed:
"     Pointer to string
"     Character position of first character to be transformed
"     Number of characters to be transformed
"     Pointer to output buffer
"     Character position of transformed string within buffer
"     Character length of output buffer
"
  ret_stat2:=STRING_TRANSFORM(LOC(in_string2),1,6,LOC(out_buffer2),
                              1,%out_buffer_len2);
"
  IF ret_stat1.isl_error_status = S'OK' AND
     ret_stat2.isl_error_status = S'OK'
    THEN
      IF out_buffer_1 < out_buffer_2
        THEN
```

```
            PUT FILE ('SYSPRINT') LIST('ABCDEF collates before
                      abcdef');
          ELSE
            IF out_buffer_1 = out_buffer_2
             THEN
               PUT FILE ('SYSPRINT') LIST('ABCDEF collates equally
                   with' & 'abcdef');
             ELSE
               PUT FILE ('SYSPRINT') LIST('ABCDEF collates after
                       abcdef');
      ELSE
        BEGIN
        IF ret_stat1.isl_severity EQ S'isl_warning'
          THEN
            PUT FILE ('SYSPRINT') LIST ('Warning ',
                      ret_stat1.isl_error_num, ' Received');
          ELSE
            IF ret_stat1.isl_severity NE S'isl_successful'
              THEN
                PUT FILE ('SYSPRINT') LIST ('Error ',
                            ret_stat1.isl_error_num, ' Received');
        IF ret_stat2.isl_severity EQ S'isl_warning'
          THEN
            PUT FILE ('SYSPRINT') LIST ('Warning ',
                      ret_stat2.isl_error_num, ' Received');
          ELSE
            IF ret_stat2.isl_severity NE S'isl_successful'
              THEN
                PUT FILE ('SYSPRINT') LIST ('Error ',
                            ret_stat2.isl_error_num, ' Received');
        END;
    TERM
```

7.3.32. STRPTIME Service Routine

Entry Points

For Basic Mode PLUS: PLS$STRPTIM

For Extended Mode PLUS: UPLS$STRPTIM

Description

STRPTIME is used to convert date and time values to a string representation based on a specified format and the locale information defined by the LC_TIME category.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$strptim'` to `'upls$strptim'`.

```
PROCEDURE STRPTIME (str_in_ptr     : WORD MACHINE POINTER,
                    str_in_start   : WORD SIGNED INTEGER,
                    str_in_length  : WORD SIGNED INTEGER,
                    fmt_str_ptr    : WORD MACHINE POINTER,
                    fmt_str_start  : WORD SIGNED INTEGER,
                    fmt_str_length : WORD SIGNED INTEGER,
                    date_time_ptr  : WORD MACHINE POINTER LOCATABLE)
          IMPORTED('pls$strptim')
          RESULT isl_error_def;
```

Parameters

str_in_ptr

is a pointer to the input character string to be converted to integer date and time values.

str_in_start

is the position in the input string where the first character to be converted is located. The first position is 1.

str_in_length

is the number of characters in the string pointed to by str_in_ptr.

fmt_str_ptr

is a pointer to the string describing the format of the input string. This format has two objects: plain characters and conversion specifications.

- Plain Characters: Plain characters are copied unchanged to the output buffer.
- Conversion Specifications: Conversion specifications consist of a % character and a terminating conversion character that determines the behavior of the specification. For details, go to the Conversion Specification Table.

fmt_str_start

is the character position in the string pointed to by fmt_str_ptr at which the first character to be used as a format directive is located. The position of the first character is 1. For details, go to the List of Directives in a Format String.

fmt_str_length

is the number of characters in the string pointed to by fmt_str_ptr.

date_time_ptr

is a pointer to the structure (of type isl_date_time_def) which will, on output, contain the date and time integer values that are converted from the string pointed to by str_in_ptr. To see this structure, go to the Date-and-Time Structures Table.

RESULT

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

STRPTIME can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'
- 51 – S'INVALID_CONVERSION_SPEC'
- 54 – S'CHARACTERS_NOT_MATCH'
- 56 – S'INVALID_DATE_TIME'

Sample Code

This code example shows how to use STRPTIME to convert date string characters to values based on locale en_US.8859-1 (English as spoken in the US). For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy elements containing the declaration of procedure STRPTIME,
" isl_error_def, and isl_date_time_def.
"
COPY ('isl-err-copy/pls', NO SOURCE);
COPY ('isl-rou-copy/pls', NO SOURCE);
"
" Declare the variables for the STRPTIME call; the return status,
" the string point, the format pointer, and the Date/Time structure.
"
DECLARE ret_stat        : isl_error_def;
```

```
DECLARE in_string       : 20 ASCII CHARACTERS LOCATABLE;
DECLARE format_string   : 12 ASCII CHARACTERS LOCATABLE;
DECLARE d_and_t         : isl_date_time_def;
"
  in_string := 'Today is 08/22/95.  ';
  format_string := 'Today is %x.';
  OPEN FILE 'SYSPRINT' STREAM OUTPUT;
  ret_stat := STRPTIME(LOC(in_string),1,20,LOC(format_string),1,12,
                   LOC(d_and_t));
"
  IF ret_stat.isl_error_status EQ S'OK'
    THEN
      PUT FILE ('SYSPRINT') LIST ('STRPTIME worked OK');
    ELSE
      IF ret_stat.isl-severity EQ S'isl_warning'
        THEN
          PUT FILE ('SYSPRINT') LIST ('WARNING ',
                   ret_stat.isl_error_num, ' RECEIVED');
        ELSE
          PUT FILE ('SYSPRINT') LIST ('ERROR ',
                   ret_stat.isl_error_num, ' RECEIVED');
                 •
                 •
                 •
```

Assuming that the locale for LC_TIME is "en_US.8859-1", the following values are calculated and placed in these d_and_t fields within the date/time structure:

```
tm_mday     22
tm_mon       7
tm_year     95
```

7.3.33. TOLOWER Service Routine

Entry Points

For Basic Mode PLUS: PLS$TOLOWER

For Extended Mode PLUS: UPLS$TOLOWER

Description

TOLOWER converts characters in a string to their corresponding lowercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of the LC_CTYPE category.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$tolower'` to `'upls$tolower'`.

```
PROCEDURE tolower(
                  string_in_ptr  : WORD MACHINE POINTER,
                  start_char_in  : WORD SIGNED INTEGER,
                  num_of_char_in : WORD SIGNED INTEGER,
                  string_out_ptr : WORD MACHINE POINTER,
                  start_char_out : WORD SIGNED INTEGER,
                  %num_of_char_out : WORD SIGNED INTEGER (LOCATABLE))
          IMPORTED('pls$tolower')
          RESULT  isl_error_def;
```

Parameters

string_in_ptr

is a pointer to the string containing characters to be converted.

start_char_in

is the character position in the string of the first character to be converted. The position of the first character in the string is 1.

num_of_char_in

is the number of characters to be converted, beginning at start_char_in.

string_out_ptr

is a pointer to the buffer in which the service routine places the converted character string. The input buffer and output buffer may be in the same buffer location.

start_char_out

is the character position in the buffer where the converted string is to begin.

num_of_char_out

Upon entry to the service routine, this integer should be set to the character length of the buffer pointed to by string_out_ptr.

If the service routine succeeds, it sets num_of_char_out to the number of characters converted.

If the service routine fails, this value is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

TOLOWER can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 17 – S'OVERLAPPING_OPERANDS'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code routine shows how to use TOLOWER to convert an input character string to lowercase alphabetic characters. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure TOLOWER.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the TOLOWER call.
"
DECLARE ret_stat         : isl_error_def;
DECLARE string_in_ptr    : 22 CHARACTERS LOCATABLE;
DECLARE string_out_ptr   : 44 CHARACTERS LOCATABLE;
DECLARE out_length       : WORD SIGNED INTEGER LOCATABLE;
"
" In this routine, the input buffer is set to the uppercase
" string 'AN UPPERCASE STRING'.  A call is made to the
" TOLOWER service routine to convert the uppercase string
" the string to lowercase and place it in string_out_ptr.
" The output buffer is twice the length of the input buffer
```

```
" in case a character converts to two characters.
"
   string_in_ptr := 'AN UPPERCASE STRING';
   out_length := 6;
   ret_stat:= tolower(LOC(string_in_ptr),1,22,LOC(string_out_ptr),1,
                  %out_length);
"
   IF ret_stat.isl_error_status EQ S'OK'
     THEN
       BEGIN
       PUT FILE ('SYSPRINT') LIST(string_out_ptr);
       END;
     ELSE
       BEGIN
       PUT FILE ('SYSPRINT') LIST(ret_stat.isl_error_num);
       END:
TERM
```

7.3.34. TOUPPER Service Routine

Entry Points

For Basic Mode PLUS: PLS$TOUPPER

For Extended Mode PLUS: UPLS$TOUPPER

Description

TOUPPER converts characters in a string to their corresponding uppercase representation. The character string must be one or more characters long. This conversion is based on the processing rules in the locale indicated by the current setting of LC_CTYPE category.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$toupper'` to `'upls$toupper'`.

```
     PROCEDURE toupper(
                    string_in_ptr : WORD MACHINE POINTER,
                    start_char_in : WORD SIGNED INTEGER,
                    num_of_char_in : WORD SIGNED INTEGER,
                    string_out_ptr : WORD MACHINE POINTER,
                    start_char_out : WORD SIGNED INTEGER,
                    %num_of_char_out : WORD SIGNED INTEGER (LOCATABLE))
            IMPORTED('pls$toupper')
            RESULT  isl_error_def;
```

Parameters

string_in_ptr

is a pointer to the string containing characters to be converted.

start_char_in

is the character position in the string of the first character to be converted. The position of the first character in the string is 1.

num_of_char_in

is the number of characters to be converted, beginning at start_char_in.

string_out_ptr

is a pointer to the buffer in which the service routine places the converted character string. The input buffer and output buffer may be in the same buffer location.

start_char_out

is the character position in the buffer where the converted string is to begin.

num_of_char_out

Upon entry to the service routine, this integer should be set to the character length of the buffer pointed to by string_out_ptr.

If the service routine succeeds, it sets num_of_char_out to the number of characters converted.

If the service routine fails, this value is undefined.

isl_error_def

is the error structure returned by the service routine (see 7.2.1).

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

TOUPPER can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 3 – S'BUFFER_TOO_SHORT'
- 17 – S'OVERLAPPING_OPERANDS'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'

- 30 – S'CORRUPTED_TABLE'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code routine shows how to use TOUPPER to convert an input character string to uppercase alphabetic characters. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure TOUPPER.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the TOUPPER call.
"
DECLARE ret_stat         : isl_error_def;
DECLARE string_in_ptr    : 22 CHARACTERS LOCATABLE;
DECLARE string_out_ptr   : 44 CHARACTERS LOCATABLE;
DECLARE out_length       : WORD SIGNED INTEGER LOCATABLE;
"
" In this routine, the input buffer is set to the lowercase
" string 'a lowercase string'.  A call is made to the
" TOUPPER service routine to convert the lowercase string
" the string to uppercase and place it in string_out_ptr.
" The output buffer is twice the length of the input buffer
" in case a character converts to two characters.
"
   string_in_ptr := 'a lowercase string';
   out_length := 6;
   ret_stat:= toupper(LOC(string_in_ptr),1,22,LOC(string_out_ptr),1,
                  %out_length);
"
   IF ret_stat.isl_error_status EQ S'OK'
     THEN
       BEGIN
       PUT FILE ('SYSPRINT') LIST(string_out_ptr);
       END;
     ELSE
       BEGIN
```

```
        PUT FILE ('SYSPRINT') LIST(ret_stat.isl_error_num);
        END;
TERM
```

7.3.35. TRANSFORM_LENGTH Service Routine

Entry Points

For Basic Mode PLUS: PLS$TRANLEN

For Extended Mode PLUS: UPLS$TRANLEN

Description

TRANSFORM_LENGTH performs these tasks:

- Calculates the maximum length of the ordering key that STRING_TRANSFORM can create for a given string length.
- Determines the maximum size that will be required for the output buffer when using STRING_TRANSFORM. The size that is returned will depend on the current setting of the LC_COLLATE category. It may be several times larger than the value that was passed.

Declaration

The I18NLIB routines rely on compile-time parameter checking. Use the appropriate ISL-ROU-COPY copy element to declare this procedure. For Extended Mode PLUS, change `'pls$tranlen'` to `'upls$tranlen'`.

```
PROCEDURE transform_length(
                          string_len : WORD SIGNED INTEGER,
                          %error_status : isl_error_def)
          IMPORTED('pls$tranlen')
          RESULT  WORD SIGNED INTEGER;
```

Parameters

string_len

is the length of the string to be transformed.

error_status

is the error structure returned by the service routine (see 7.2.1).

RESULT

is the maximum byte length required by STRING_TRANSFORM to hold a transformed string of the length specified by string_len. If error_status is not OK, RESULT is undefined.

Error Statuses

Refer to Appendix A for a detailed description of the error statuses returned.

TRANSFORM_LENGTH can return the following error statuses in the *isl_error_def* field:

- 0 – S'OK'
- 20 – S'EV_NAME_INVALID_CHAR'
- 21 – S'EV_VALUE_INVALID_CHAR'
- 26 – S'LOCALE_NOT_FOUND'
- 34 – S'INVALID_BM_ADDRESS'
- 35 – S'INVALID_EM_ENVIRONMENT'
- 39 – S'GET_ENV_ERROR'
- 41 – S'ASTORE_READ_ERROR'
- 42 – S'ASTORE_WRITE_ERROR'
- 43 – S'CREATE_BANK_ERROR'
- 44 – S'CHECKPOINT_RESTART_ERROR'

Sample Code

This code routine shows how to use TRANSFORM_LENGTH to determine the length of the output buffer returned by STRING_TRANSFORM for a specific length string. For an Extended Mode PLUS application, change the copy element from `'isl-rou-copy/pls'` to `'isl-rou-copy/upls'`.

```
MODULE;
"
" Copy definition of the error status that is returned; then
" copy element containing the declaration of procedure
" TRANSFORM_LENGTH.
"
COPY ('isl-err-copy/pls',NO SOURCE);
COPY ('isl-rou-copy/pls',NO SOURCE);
"
" Declare variables for the TRANSFORM_LENGTH call.
"
DECLARE ret_stat         : isl_error_def;
DECLARE string_len     : WORD SIGNED INTEGER;
DECLARE xformed_len  : WORD SIGNED INTEGER LOCATABLE;
DECLARE in_string       : 6 CHARACTERS LOCATABLE;
DECLARE out_buf(*)     : 1 CHARACTER BASED LOCATABLE;
DECLARE area1            : 1000 WORDS AREA STATIC:=NULL;
DECLARE out_ptr         : WORD MACHINE POINTER;
"
  in_string := 'ABCDEF';
  string_len := 6;
"
" Call TRANSFORM_LENGTH to determine the maximum
```

```
" transformed length a string of 6 characters could produce.
"
   xformed_len := transform_length(string_len, %ret_stat);
   IF ret_stat.isl_error_status NE S'OK'
     THEN
       BEGIN
         .
         .                       " error occurred
         .
       END;
"
" Allocate a buffer large enough to hold transformed string.
"
   ALLOCATE out_buf BOUND(out_buf,xformed_len) IN area1 SETTING
      out_ptr;
"
"  Call STRING_TRANSFORM.
"
   ret_stat := string_transform(LOC(in_string),1,6,out_ptr,
                   1,%xformed_len);
TERM
```

Section 8
Products with I18N Features and Capabilities

This section describes I18N functionality implemented in products other than I18NLIB and the language processors. Many of these products reference the I18NLIB service routines to implement the I18N functionality. The information provided in this manual is a basic description of the functionality not a detailed description. If more information is necessary, please refer to the appropriate product documentation.

8.1. Business Information Server

Note: *The Business Information Server was formerly known as MAPPER, which remains the installation and support name.*

Internationalized Business Information Server

- Includes enhanced ELT and RET functions.
- Uses coded character set identifiers (CCS ID).
- Includes 8-bit-transparent terminal interface.

Enhancements to ELT and RET Functions

The Element (ELT) and Retrieve (RET) functions have been enhanced to enable an application to specify or detect coded character set identifiers (CCS ID) in an OS 2200 file.

Applications and CCS IDs

Internationalized Business Information Server applications use CCS-IDs for both data announcement and data integrity. The application, not the system, tags the data.

Terminals and CCS IDs

Octal values 040 through 0377 pass cleanly through the Business Information Server terminal interface, EXCEPT for octal value 0177, which might not pass unmodified.

The TCCS$ reserved word indicates the CCS ID of the associated terminal. This reserved word can be used to tag data at the application level.

For More Information

For additional information on migrating the Business Information Server to an I18N environment see Section 2.

8.2. Communications Management System (CMS 1100)

Internationalized CMS 1100

- Transmits 7-bit and 8-bit characters.
- Supports the Unisys LT300 terminal.
- Identifies the coded character sets (CCS) of incoming terminal messages and passes this information to either MCB or the Exec.
- Initializes the new P$CCS field, which is supported by MCB. This field identifies the CCS in use.
- Provides special terminal handlers to convert between 7-bit and 8-bit character representations.

I18N Implications for CMS Interfaces

The user interfaces (GASIF, TSAM, and COMPOOL) are not affected. They continue to pass data to the user unchanged and without modification by CMS. CMS has no knowledge of what is contained in the user portion of any input or output text, including the character set in use.

8.3. Display Processing System

Note: *The Display Processing System was formerly known as DPS 2200, which remains the installation and support name.*

Internationalized Display Processing System

- Runtime input/output functions allow throughput of 8-bit CCSs.
- Handles 8-bit characters for the Unisys LT300 terminal and INFOCONNECT.
- Enables users to define forms that use an ISO 8859 character set.
- Uses I18NLIB service routines to process characters, strings, and coded character sets (CCS).
- Includes a new form level attribute.
- Can activate and deactivate I18N code.

Screen Handling Features

The I18N Display Processing System can be used to manage screen-handling for the Unisys LT300 terminal, as well as the other 8-bit terminals supported. For more about using enhanced Unisys terminals, see Migrating Terminals.

Screen-handling operations are transparent to an I18N application, but this application must generate the message text for the Display Processing System in an 8-bit CCS appropriate for the terminal.

Use of I18N Service Routines

The I18N Display Processing System uses the I18NLIB service routines to

- Perform string and character classification, comparison, and conversion.
- Transliterate literal text in a screen from one CCS to another.
- Enable (1) a screen designer to specify the CCS for literal text on the screen and (2) an application to transliterate this literal text before displaying it on the terminal.

New Form Level Attribute

The FORMGN, FLDP and FLMU utilities have a new form level attribute to specify the CCS of the literal text in the form.

Ability to Activate/Deactivate I18N Code

The I18N Display Processing System includes a new build flag, so that sites can use Display Processing System building and configuring mechanisms to turn the I18N code on or off.

- When the I18N code is ON, the I18NLIB service routines can be used.
- When the I18N code is OFF, transliteration is unsupported.
- When the I18N code is ON or OFF, the GETENV service routine can be used to determine the default CCS for literal text in a screen.

8.3.1. Changes to Existing Modules

The following code for	Has been changed to call these I18N service routines
DEF-FRM-SUBS/CODE	ISALNUM, ISALPHA, TOLOWER, and TOUPPER
FIELDS/CODE	ISDIGIT and ISALNUM
FLMU-GENWS/CODE	ISLOWER, TOUPPER, ISDIGIT, and ISALNUM
GET/CODE	ISLOWER, TOUPPER, ISDIGIT, and ISALPHA
READ/CODE	ISLOWER, TOUPPER, ISALPHA, and ISDIGIT
SEND/CODE	CCS_TO_CCS

8.3.2. Changes to Initialization Functions

Initializing the I18N Display Processing System

The initialization functions for the I18N Display Processing System include the following: D$INIT, D$INIT1, D$INIT2, D$INIT3, and D$INIT4.

These functions use the SETLOCALE service routines instead of the internal versions of the Display Processing System to establish the locale for the current activity. This information is stored to be used by other Display Processing System functions later.

Reinitializing the I18N Display Processing System

If you change the locale or the coded character set (CCS) after initialization, you will need to reinitialize the Display Processing System for the change to take effect.

8.3.3. Changes to Input Functions

Processing Input with the I18N Display Processing System

The input functions for the I18N Display Processing System include the following: D$GETFL, D$GETFL1, D$GETLI, D$GETLI1, D$GTSCN, D$GTSCN1, and D$READ.

These functions use the I18NLIB service routines instead of the internal functions of the Display Processing System to verify, convert, or compare any user-entered data.

Input functions for the I18N Display Processing System perform these operations	By using these I18NLIB service routines
Verify that data is of the correct character class	Character class (ISA) service routines
Convert strings or characters to lowercase	TOLOWER
Convert strings or characters to uppercase	TOUPPER
Compare strings or characters	STRING_COLLATE

Processing Constraints

The input functions for the I18N Display Processing System apply to user-entered data only, not to any file element names or Display Processing System commands and keywords.

8.3.4. Changes to Output Functions

Processing Output with the I18N Display Processing System

The output functions for the I18N Display Processing System include the following: D$RESEND, D$SEND, D$SEND1, D$SENDF, and D$SENDF1.

These functions compare the current coded character set (CCS) to the defined CCS for the screen before displaying the screen on the terminal.

If the current CCS differs from the defined CCS, the CCS_TO_CCS service routine is used to transliterate the literal text on the screen to the current CCS.

Output Function Not Affected

The I18N features of the Display Processing System do not affect the D$PTSCN function.

8.3.5. Changes to Copy Procedures

Copy procedures for the Display Processing System are included in the user's application transactions. The DEF-ATTRIBUTES procs are the only ones affected by I18N features.

Change to DEF-ATTRS/UCOBP (COBOL Compiler)

In the attribute definitions for the COBOL compiler, this line is added before the D-ATT-HIGHLT-BLINK attribute.

```
05 D-ATT-I18N_CCSNUM PIC 9(5) COMP VALUE 69.
```

Change to DEF-ATTR-DEC/H (C Compiler)

In the attribute definitions for the C compiler, this line is added before the D-ATT-HIGHLT-BLINK attribute.

```
#define D_ATT_CCS_NUM 69
```

8.4. Element Processor (ELT)

Internationalized ELT

- Can handle all legal coded character sets (CCS).
- Inserts elements into a program file.
- Updates existing symbolic elements in a program file.

Use of P, Q, and I Options

The P, Q, and I options can be used on the ELT call statement. The rules for governing these options are the same as for SYSLIB SIR$.

See *General Rules for P and Q Options (SYSLIB SIR$)*

See *Special Rules for I Option (SYSLIB SIR$)*

Handling Control and Editing Statements

Internationalized ELT accepts the following statements from both runstreams and files:

- Change control statements
- Partial line change control statements
- Line change editing statements

See *Rules for New Runstream Images (SYSLIB SIR$)*

8.5. Enterprise Output Manager

Note: *The Enterprise Output Manager was formerly known as DEPCON, which remains the installation and support name.*

Internationalized Enterprise Output Manager

- Uses coded character set (CCS) tables to determine what action to take for a given CCS.
- Supports translation tables that map between file CCSs and printer CCSs.
- Recognizes CCS IDs in print files.

8.6. Exec ASTORE Feature

ASTORE and I18NLIB

The I18NLIB service routines use the Exec ASTORE feature to maintain category information. I18NLIB uses CREATE$BANK to acquire a data bank to store category information, and it stores the virtual address (VA) of the bank in ASTORE.

ASTORE and Security Implications

ASTORE maintains information at the activity level. When an activity terminates, ASTORE data and the I18NLIB data bank are no longer accessible.

ASTORE and Recovery Operations

If your system should fail, the ASTORE information is not recoverable. Since ASTORE points to the I18NLIB data bank, the data bank is also no longer accessible to the I18NLIB service routines.

8.7. Extended Language Message System (ELMS)

Internationalized ELMS

I18N ELMS enables a site to

- Define the written language attributes specific to its locale.
- Determine the coded character set or sets (CCS) needed at the site.
- Build and access CCS-distinct message module data banks (MDB).

8.7.1. Defining Messages

Procedure

Use a source element to enter an ELMS message definition for an application. You can then use this element to create the message module data bank (MDB).

Format

ccc/*l*[*lll*][-BASE]

Values

ccc

is the unique component identifier (3 characters from A through V and 0 through 9).

/[///]

is the natural language identifier (1 alphabetic character and up to 4 alphanumeric characters).

-BASE

identifies the base language for the application (uppercase or lowercase).

Code Examples for Defining Messages

Example 1

This example illustrates how to specify English as the base language for component ABC.

```
ABC/ENG-BASE
```

Example 2

This example illustrates how to specify English as the base language element for component ABC with German, Spanish, and French as additional language identifiers.

```
ABC/ENG-BASE
ABC/GER
ABC/SPA
ABC/FRE
Procedures
```

8.7.2. Creating Message Module Data Banks (MDB)

Procedures

Use the PROCESS statement with the ELMS/SKELETON SGS to specify a message element for an MDB. There should be an MDB element for every locale in use at your site or subsidiary. Make sure that the MDB name reflects the name of the locale and the name of the coded character set (CCS), if any.

Format

```
PROCESS source-element
```

where *source-element* is the name of the ELMS source element in the standard format

```
ccc/l[lll][-BASE].
```

The **ccs** field is added to specify a version name for the output MDB. This field is the coded character set to be used for the messages.

```
ccc/l[lll][-BASE]ccs
```

Code Examples for Creating MDBs

Example 1

If the SOLAR ELMS SGS specifies a language/CCS combination "en_US.8859-1/ISO8859-1", the PROCESS statement would read

```
PROCESS ABC/ENUS ISO8859-1
```

which produces the output element ABCENUS$MDB/ISO8859-1.

Example 2

If the SOLAR ELMS SGS specifies a language/CCS combination "en_US.8859-1/8859-1", the PROCESS statement would read

```
PROCESS ABC/ENUS 8859-1
```

which produces the output element ABCENUS$MDB/8859-1.

Example 3

If the SOLAR ELMS SGS specifies a language/CCS combination "en_US.8859-1/CCS-A", the PROCESS statement would read

```
PROCESS ABC/ENUS CCS-A
```

which produces the output element ABCENUS$MDB/CCS-A.

8.7.3. Creating Local-Language Message Modules (MDB)

Background

The Local-Language Processor (LLP) is used to localize 2200 programs and applications. If your site or subsidiary handles more than one natural language, use this processor to prepare user messages for multiple locales and coded character sets (CCS).

LLP Processor Call

```
@LLP[,options]
```

where *options* include both internal options and feature options. If you do not specify any options, the LLP Main Menu appears (see Related Topics).

LLP Internal Options

B – Trace input requests and save data when terminating.

O – Display available options.

T – Prompt for list of languages to use when displaying LLP menus.

V – Prompt for string to use for canceling current request (initial setting = CANCEL).

W – Display element name created by LLP when new element is created.

X – Exclusively assign files before batch mode execution.

Y – Display element name used by LLP to read data and process request.

LLP Feature Options

A – Create the MDB for all language elements in the product's file.

E – Provide a message text for local translations.

F – List message numbers new to a new product release.

G – List message numbers with new text.

H – Move unchanged message text to a new release.

I – Create a local attribute element.

J – Move local attributes from last release to new release (regardless of changes in base element).

K – Move local attributes from last release to new release (no changes in released attributes).

M – Move local attributes from last release to new release. (If base elements disagree, select the one to use from the list that appears.)

Processing Message Attributes (LLP)

To process an ELMS message attribute, follow these steps:

1 From the LLP Main Menu, select the **Process Attributes** option to either (1) create local attributes or (2) move the attributes to a new product release level.

2 When the LLP Attributes Menu appears, select one of the following options:

- **Create a Local Attribute Element** to display a series of prompts for creating or updating the attributes of a specific product's messages.
- **Move Local Attributes to a New Release** to display another menu for moving or upgrading local attributes to a new product release.

Creating or Updating Attribute Elements (LLP)

Step 1: Select an Element

- If this is the first request to modify an attribute, LLP creates a special element with a version of /ATTRILOCAL in the product's file.
- If this not the first request, the ATTRILOCAL (local) element already exists. You are asked if LLP should use the local version or the Unisys base version of attributes for the change.

Step 2: Select Messages to Modify

- If you know the message number, enter it at the prompt.
- If you do not know the message number, press the transmit key to display the messages sequentially.
- If you want to modify more than one message, use the transmit key to step through the messages one at a time.

Step 3: Modify Attributes of a Message

1. When LLP displays the attributes that can be changed, select the one to be changed, enter the desired value, and transmit the line.
2. When you have changed all attributes to the desired value, transmit from the SOE character to commit the changes.

Step 4: Commit Changes

1. When you have entered all changes, enter a minus sign (-).
2. When the options for saving or ignoring the changes appear, enter the desired option number at the SOE character.

Moving or Upgrading Local Attributes (LLP)

Determine the desired result, then select the appropriate option.

Decide Which Attribute to Move

Compare a previous product release (old release) with the new release. If a difference occurs, select the source of the new attribute value (old release, new release, or local changes from old release).

Keep Local Attributes

Retain the local attribute values of the previous product release (even if the values in the base element have changed).

Update Local Attributes

Move local attributes to the new product release (only if there are no changes to the attribute values in the base element).

Processing Message Text (LLP)

To process ELMS message text, follow these steps:

1. From the LLP Main Menu, select the Process Text option to either (1) create a local text element or (2) move the text elements to a new product release level.
2. When the LLP Text Menu appears, select one of the following options:
   - **Provide message text for editing a local element** to create a local language element that can be used to customize message text.

- **Move local translated element to a new release** to move the local-language text from a previous product release to a new release

Customizing Message Text (LLP)

CAUTION:

Do not modify the -BASE or -SUB elements, because the LLP uses them for a release upgrade. If they are changed, the LLP cannot correctly update a site or subsidiary's modifications to a new product release.

Step 1: Specify product files to be modified.

Use of these formats to supply the qualifier*filename, product identifier, and local language.

If you use this format	LLP displays:	And you need to
qualifier*file.product/language	No additional prompts	Do nothing (see step 2).
qualifier*filename	Additional prompts	Transmit to retain current file. Enter ? for help.
qualifier*file./language	List of valid products in the file	Select a product from the list.

Step 2: LLP creates an output element.

This element contains the version name of the 3-character language code and dialect (if specified).

Step 3: View or edit this element.

Use any editor to read, update (translate), or create additional dialects from this element.

Moving Local-Language Text (LLP)

Determine the desired result, then select the appropriate option.

- List Numbers for New and Changed-Text Messages

  List those messages with text changes between product releases.

- List Numbers for New Messages Only

  List messages that are in the new release, but not in the previous release.

- Copy Local Text where New Release Unchanged

  Move the local translated text to a new output file in a new release if the text in the source element did not change between releases.

Creating New MDB Elements (LLP)

To create new MDB elements, follow these steps:

1. From the LLP Main Menu, select the Create New ELMS MDB Elements option to (1) process local text translations and local attributes, and (2) make them available to install with a product.
2. When the Create Message Data Base Element(s) Menu appears, select one of the following options:
   - **Process All Languages for a Component** to process all languages and message text elements within the product file for a specified product. Use this option when you have created local attributes.
   - **Process One Language for a Component** to process one local-language message text element within the product file for a specified product. Use this option only when you have created and then translated a new message text element, not if you have created or changed attribute elements.

Displaying Message Elements (LLP)

To display ELMS message elements, follow these steps:

1. From the LLP Main Menu, select the List ELMS Base Source Elements option to display the message elements in a product's file.
2. The LLP then displays all elements with a version name containing -BASE. This list includes the product identifier and the language of the base element. For example, the ELMS list is

```
NLA/ENG-BASE
NLB/ENG-BASE
NLM/ENG-BASE
NLN/ENG-BASE
```

8.7.4. Testing Messages (ETEST)

Background

The ELMS utility processor ETEST displays the messages in registered or unregistered message modules (MDB). You can use ETEST to review a newly created MDB before you install and register it.

Using the Language Field on the Calling Sequence

If you specify the *filename* field on the @ETEST processor call sequence, you must also specify the *language* field to indicate the language of, or locale corresponding to, the MDB you want to use for message retrieval and display.

The *language* field has two formats.

```
language[dialect]
```

specifies the language and dialect version (if any) of the MDB. This format is included for compatibility with earlier versions of ELMS.

```
locale[/ccs-id]
```

specifies the locale and optional coded character set (CCS) of the MDB.

Code Examples (ETEST)

Example 1

This example illustrates a request to ELMS to search for an MDB corresponding to a specific locale for product ABC.

```
@ETEST ABC,1,,EN-US.8859-1
```

Example 2

This example illustrates a request to ELMS to search for an MDB corresponding to a specific locale and a specific CCS for product ABC.

```
@ETEST ABC,1,,EN-US.8859-1/CCS-A
```

Error Messages (ETEST and EHELP)

Following are the I18N-related error messages for the ETEST and EHELP processors. All ETEST and EHELP messages are stored in the NLN component.

Condition Identifier	Message Code Name and Text
*ERROR NLN000006	Invalid language mnemonic specified on processor call.
*ERROR NLN000022	If the filename field is specified, the language[dialect] fields must also be specified.
*ERROR NLN000024	If the explanation-level or language[dialect] fields are specified, a message identifier must also be specified.

8.7.5. Displaying Message Explanation Levels (EHELP)

Background

The ELMS utility processor EHELP displays additional explanation levels for a message from the message identifier returned by ELMS. You can use EHELP to redisplay one message or a range of messages from a registered message module (MDB).

Using the Language Field on the Calling Sequence

If you specify the *filename* field on the @ETEST processor call sequence, you must also specify the *language* field to indicate the language of, or locale corresponding to, the MDB you want to use for message retrieval and display.

The *language* field has two formats.

```
language[dialect]
```

specifies the language and dialect version (if any) of the MDB. This format is included for compatibility with earlier versions of ELMS.

```
locale[/ccs-id]
```

specifies the locale and optional coded character set (CCS) of the MDB.

Code Examples (EHELP)

Example 1

This example illustrates a request to ELMS to search for an MDB corresponding to a specific locale for product ABC.

```
@EHELP ABC,1,,EN-US.8859-1
```

Example 2

This example illustrates a request to ELMS to search for an MDB corresponding to a specific locale and a specific CCS for product ABC.

```
@EHELP ABC,1,,EN-US.8859-1/CCS-A
```

ELMS Error Messages

See the preceding topic *ELMS: Error Messages (ETEST and EHELP)*

8.7.6. Using Locales and CCSs in Message Delivery

See 1.6.4.

8.7.7. Requesting Message Delivery in a Specific Language

Requesting Message Delivery

To request delivery of an ELMS message, an application calls ELMS and supplies the location of a request packet, a result buffer, and an insert buffer list. The request packet contains input data such as the condition identifier, message type, and message routing specification.

Responding to Message Delivery Requests

For each request, ELM searches for the message data bank corresponding to the language identifier, which is specified in the language field of the request packet. This

field is set by the ELMS caller using the application's own facilities or, in some cases, a POSIX environment variable.

Result if Language Identifier is Specified

If a language identifier is specified, but ELMS cannot find the corresponding message data bank, ELMS terminates the process and returns an error to the caller.

Result if Language Identifier is Not Specified

- If a language identifier is not specified, ELMS searches for a language specification in the value of the LANG environment variable, first in the run-level (user) profile for the session, and then in the default system-level profile.
- If the language identifier is still not found, ELMS then searches the component's designated base language (-BASE).

Specifying Request Packet Fields

Following are the fields that can be specified within the format of the request packet.

language_code

Specifies the identifier for the language whose message module is to be used; includes an optional one-character dialect identifier.

start_line_cnt

Used for messages that are too long for ELMS to retrieve with a single retrieval request. On the first request, the application should set the value to zero.

If the buffer is too small, ELMS sets the field to nonzero and then sets the language code, dialect, and explanation level. (This ensures retrieval of the correct message version on the next request.)

result_buffer_length

Specifies the length in words of the result buffer. The maximum size is 62 words, which is the maximum size of a message with embedded kanji (Japanese) characters.

locale_pref

Specifies the locale that determines the first message module to be searched for a message. This optional field can be up to 16 characters long. Use trailing zeros for shorter locale names.

locale_used

Specifies the locale name corresponding to the message module from which a message is delivered successfully. This name may or may not be the same as the one in *locale_pref*.

CCS_pref

Specifies the CCS that determines the first message module to be searched for a message. This optional field can be up to 16 characters long. Use trailing zeros for shorter CCS names.

CCS_used

Specifies the CCS name corresponding to the message module from which a message is delivered successfully. This name may or may not be the same as the one in *CCS_pref.*

Using the Text Attribute Table

Identifying Text Attributes

The text attribute table identifies various text attributes within the ELMS result buffer and the ELMS insert buffer.

This table can be used to identify which characters are kanji (Japanese) characters and which are ASCII characters.

Determining Word Length and Format

The *att_tbl_len* field contains the number of words in the text attribute table. These words are in the SYSLIB SAR$ internal format.

Using Multiple Words

For message response information in the result buffer, the text attribute table can contain multiple words to specify multiple text attributes.

Using the Table Attribute Length Field

In the result buffer, the value of the *att_tbl_len* field is used for message response information, message line responses, and free-format message responses.

In the insert buffer, it is used for the insert string.

A value of zero in this field indicates that there is no text attribute table. A nonzero value determines the placement of the table within the two buffers.

For more information about the SYSLIB SAR$ internal format, refer to *SYSLIB Library Programming Reference Manual*.

8.7.8. Setting Environment Variables (EPROF)

Background

The EPROF processor

- Accepts any locale name that conforms to XPG format, even if, due to length or character validity, this name does not conform to OS 2200 file-naming conventions.

- Supports the environment variables LANG, EXPL, LC_ALL, LC_COLLATE, LC_CTYPE, LC_DATE, and LC_MONETARY.

Processor Call Format

```
@SYS$LIB*ELMS.EPROF [,{R|I}]
      variable_name,variable_value [,profile_type_keyword]
```

Values

R

is the instruction to retrieve the current value of an environment variable.

I

begins an interactive dialog so that you can specify environment names and values.

variable_name

is the name of the variable that you want to change.

variable_value

is the new value for the variable.

profile_type_keyword (optional)

specifies either USER (default) or SYSTEM (set by security administrator).

Code Examples (EPROF)

Example 1

This example illustrates how to set the value of LANG in the user profile to German.

```
@EPROF LANG,GER

User environment variable is now 'GER'
```

Example 2

This example illustrates how to display the value of LANG in the system profile.

```
@EPROF LANG,GER
User environment variable is currently 'GER'
```

Example 3

This example illustrates how to use the interactive dialog feature to set the value of LC_MESSAGES in the user profile to US English.

```
@EPROF,I
Beginning dialog to access environment variables.
```

```
Enter name of environment variable to modify:  >LC_MESSAGES
Enter locale name:  >en_US.8859-1
User environment variable LC_MESSAGES is now 'en_US.8859-1'.
Enter name of environment variable to modify:  >exit
END ELMS EPROF UTILITY.
```

8.7.9. ETEST and EHELP Processor Options

Background

Both the ETEST and the EHELP processors

- Include calls to indicate the language or locale of the ELMS message module data bank (MDB).
- Accept only those locale names that conform to OS 2200 file-naming conventions.

ETEST Processor Call Format

```
@ETEST, [options] component-identifier,;
                  message-number[s];
                      [, explanation-levels;]
                         [, language [dialect];]
                             [, filename;]
                                [, insert-option]
```

EHELP Processor Call Format

```
@EHELP, options message-identifier;
                      [, explanation-level];
                            [, language [dialect]  ]
```

Format and Values of Language Options

In both these processor call formats, *language* is the language or locale of the MDB to be used for message retrieval and display.

Format 1

language[**dialect**]

indicates the language and dialect (if any) of the MDB. This value is provided for compatibility with older versions of ELMS.

Format 2

locale[**ccs-id**]

indicates the locale and optional coded character set (CCS) of the MDB.

Procedure

Specify an ELMS SGS for each product component as well as the SOLAR SGSs for installing the product.

Format

```
ELMS ;
   MODE,mode-list ;
   FILE,filename ;
   COMPONENT,component-id ;
   LANGUAGE,language[,language] ;
   [DISPLAY,display-id]
```

LANGUAGE Identifier

LANGUAGE is the language identifier or locale related to a particular message element. The -BASE element suffix must be included with the default language.

The *language*[,*language*] formats can be either *lang-locale* or *lang-locale/ccs*.

lang-locale

specifies a natural language or a locale.

lang-locale/ccs

specifies a language or locale and a coded character set (CCS).

You can specify a maximum of 16 characters for either *lang-locale* or *ccs*.

Examples

```
ELMS ;
   MODE,A ;
   FILE,filename ;
   COMPONENT,CO1 ;
   LANGUAGE,ENG-BASE,GER,SPA, GER, SPA, en-US, de-DE.8859-1

ELMS ;
   MODE,A ;
   FILE,filename ;
   COMPONENT,CO2 ;
   LANGUAGE,ENG-BASE ;
   DISPLAY ETEST
```

8.7.10. ELMS Error Messages

Following are the I18N-related error messages for the ELMS message delivery system (MDS). All MDS messages are stored in the NLM component. The message code names are from the ERRDEFS element, and the message text is derived from the NLM-BASE message module.

Table 8–1. ELMS Message Delivery System Error Messages

Condition Identifier	Message Code Name and Text
NLM000046	MDS_TOO_MANY_LINES The text for message <insert1> <space1> contains too many lines to be retrieved. The maximum is <insert_dec2>.
NLM000100	MDS_INCOMPATIBLE_MDB Message message-number is stored in a message database that is not compatible with the current ELMS delivery system.
NLM000200	MDS_ALTERNATE_LANGUAGE_USED An alternate language was used because an ELMS condition condition-identifier was generated seeking at least one profile-specified candidate.
NLM000202	MDS_NO_PRODUCT_ID_HARD The product specified by condition condition-identifier is not registered with ELMS. A hard-coded English message was delivered due to errors referencing the ELMS message database. This message is hard-coded and cannot be translated.
NLM000204	MDS_NO_BANK_MDS_HARD The product specified by condition condition-identifier is registered, but the common-banked message is not accessible. A hard-coded English message was delivered due to errors referencing the ELMS message database. This message is hard-coded and cannot be translated.
NLM000205	MDS_SPEC_LANG_UNAVAILABLE The message <insert1> <space1> is not available in the specified language.
NLM000206	MDS_LANG_NOT_IN_DIR The language specified for delivery of message <insert1> <space1> is not registered in this system's ELMS Directory.
NLM000210	MDS_MESSAGE_NOT_FOUND_HARD The message message-number cannot be found in the product database. A hard-coded English message was delivered due to errors in referencing the ELMS message database. This message is hard-coded and cannot be translated.
NLM000223	MDS_TOO_MANY_ATTRS The version of message message-number specified in the request packet contained too many attributes on one line. ELMS substituted a different language version.
NLM000301	MDS_GETENV_ERROR The Exec GET$ENV profile access interface returned error status <insert1> <space1>.
NLM000320	MDS_I18N_DIALECT_SPEC_ERROR The ELMS request packet specification of language was ambiguous, because both the 4th character of the language code, used for a dialect digit in 2R2 and prior, and the dialect field defined in a previously unused location for 3R1, are set. The 4-character contiguous code will be used for language.

Table 8–1. ELMS Message Delivery System Error Messages

Condition Identifier	Message Code Name and Text
NLM000325	MDS_I18N_EV_TOO_LONG The locale-name value <locale-name> for a message-related Environment Variable exceeded the I18N limit of 16 characters. That value was not used in seeking language candidates.
NLN000027	The locale-form language parameter is too long (>16 characters total).

8.8. FORTRAN Compiler

8.8.1. I18NLIB Service Routines Callable from the FORTRAN Compiler

Accessing Service Routines through the C and COBOL Compilers

The FORTRAN Compiler has no language bindings to the I18NLIB service routines. To access these routines, a FORTRAN program must do one of the following:

- Call the COBOL Compiler entry point.
- Make an interlanguage CALL to a program in C or COBOL.

For either access method to work, the FORTRAN Compiler data type must match the COBOL Compiler data type that the service routine is expecting.

8.8.2. Using the C Compiler to Access the I18NLIB Service Routines

If an interlanguage call to the C Compiler is used, the C program accesses the I18NLIB service routines through the C Compiler entry point names. The following example illustrates how a FORTRAN program would call a C program to determine if a character is alphabetic.

FORTRAN Program

```
PROGRAM TEST
INTEGER C_RTN
IF (C_RTN ('a').NE.O) PRINT *, 'IS ALPHABETIC'
END
```

C Program

```
#include <ctype.h>
# pragma interface C_RTN
int C_RTN (char*p)
}
return (isalpha (*p));
}
```

8.8.3. Using the COBOL Compiler to Access the I18NLIB Service Routines

Using Standard Calling Sequence

Code generated from the FORTRAN and COBOL Compilers share the same standard calling sequences. If this sequence is used, calls can be made directly to the COBOL service routines. The FORTRAN program must use the same service routine name as the COBOL program does.

Sample Calling Sequence

The following example illustrates a call to the COBOL SETLOCALE service routine.

```
CALL UCOB$SETLOC (isl_status, 'LC_ALL', '_DEFAULT', resulting_locale,
resulting_locale_len)
```

Note that *isl_status* is a one-word integer that is filled in during the execution of the service routine. For a detailed layout of this record, see COBOL error structure.

Using Corresponding Data Types

The COBOL Compiler interfaces to the I18NLIB service routines use integer and data types. When calling these routines from a FORTRAN program, you need to use the corresponding FORTRAN data type.

This COBOL data type	Corresponds to this FORTRAN data type
PIC X (n)	CHARACTER*n
PIC 9 (10) BINARY	INTEGER

8.8.4. Sample Code: FORTRAN Compiler to COBOL Compiler Entry Point

This code example shows how a FORTRAN main program calls the COBOL entry point for the I18NLIB SETLOCALE and STRING_COLLATE service routines.

```
      PROGRAM MAIN
C
C     Define the following variables:
C        ISL_STATUS                  - the ISL status returned by
C                                      the I18NLIB service routine
C        RESULTING_LOCALE            - the name of the resulting
```

```
C                                       category value
C       RESULTING_LOCALE_LEN          - the length of the name of
C                                       the resulting category value
C       STRING1                       - the first string to compare
C       STRING2                       - the second string to compare
C       COLLATION_RESULT              - the result of the comparison
C
      INTEGER       ISL_STATUS,RESULTING_LOCALE_LEN, COLLATION_RESULT
      CHARACTER*16  RESULTING_LOCALE,STRING1,STRING2
C
C     Call SETLOCALE to initialize all categories to their default
C     values.
C
      CALL UCOB$SETLOC (ISL_STATUS, 'LC_ALL', '_DEFAULT',
     *     RESULTING_LOCALE, RESULTING_LOCALE_LEN)
      IF (BITS (ISL_STATUS,1,3).NE.0) THEN
         PRINT *,'Service failed with error;', BITS (ISL_STATUS,19,18)
         STOP
      ENDIF
C
C     If the locale for LC_COLLATE is not French, set it to French.
C
      IF (RESULTING_LOCALE .NE. 'fr_FR.8859-1') THEN
      CALL UCOB$SETLOC (ISL_STATUS, 'LC_COLLATE', 'fr_FR.8859-1',
     *     RESULTING_LOCALE, RESULTING_LOCALE_LEN)
      IF (BITS (ISL_STATUS,1,3).NE.0) THEN
         PRINT *,'Service failed with error;', BITS (ISL_STATUS,19,18)
         STOP
         ENDIF
      ENDIF
C
C     Establish two strings and call the STRING_COLLATE service
C     routine to compare them.
C
      STRING1 = 'Côte'
      STRING2 = 'Côté'
C
      CALL UCOB$STRCOLL (ISL_STATUS, STRING1, STRING2,
COLLATION_RESULT)
      IF (BITS (ISL_STATUS,1,3).NE.0) THEN
         PRINT *,'Service failed with error;', BITS (ISL_STATUS,19,18)
      STOP
      ENDIF
C
      IF (COLLATION_RESULT .EQ. -1) THEN
      PRINT *,'String 1 sorts before String 2'
      ENDIF
      IF (COLLATION_RESULT .EQ. 0) THEN
      PRINT *,'String 1 is equal to String 2'
      ENDIF
      IF (COLLATION_RESULT .EQ. 1) THEN
      PRINT *,'String 1 sorts after String 2'
```

```
      ENDIF
C
      END
```

8.8.5. Error Structure and Error Statuses

The FORTRAN Compiler uses the error structure of the COBOL Compilers.

The error statuses returned from the FORTRAN Compiler service routines match the error statuses returned in the language bindings of the COBOL Compiler.

For more information, refer to the error status page for each of the corresponding service routines callable from these compilers.

8.9. General Syntax Analyzer (GSA 1100)

Internationalized GSA 1100

- Is 8-bit transparent.
- Works with any coded character set (CCS) that is ASCII-like.
- Can handle 8-bit data when the CHARSET constant of the Input Routine configuration is set to ISO.

Note: *The TCON processor does not allow 8-bit characters to be used in literals or comments.*

Uppercasing ASCII Characters

Use a lexical specification with the SHIFT operator (!) to indicate the uppercasing of the ASCII characters a through z and A through Z.

Uppercasing Characters With Diacritical Marks

Use the TOUPPER service routine with your application to indicate the uppercasing of characters with diacritical marks.

8.10. Interactive Processing Facility

Note: *The Interactive Processing Facility was formerly known as IPF 1100, which remains the installation and support name.*

Internationalized Interactive Processing Facility

- Supports 7-bit and 8-bit character sets.
- Includes a dynamic I18N Feature.
- Has an enhanced OLD command.
- Has four new system variables.
- Has four enhanced system functions.

Support for 7-Bit and 8-Bit Character Sets

The I18N Interactive Processing Facility supports text editing operations for 7-bit and 8-bit character sets including US ASCII.

Dynamic I18N Feature

A system administrator can use the COMUS CONFIGURE command to dynamically activate or deactivate the I18N Feature without having to reinstall the Interactive Processing Facility.

See *Interactive Processing Facility: Configuring, Activating, and Deactivating the I18N Feature*

Enhanced OLD Command

The optional keyword parameter CCS has been added to specify the coded character set (CCS) to be used for the editing object data after the OLD command has been executed.

New System Variables

$CCS – Contains the value of the CCS to be used by the Interactive Processing Facility workspace or lookspace.

$TERMINALCCS (also shown as $TCCS or $TERMCCS) – Contains the value of the CCS for a user's terminal.

$COLLATE – Contains the name of a locale that specifies the character comparison conventions to be used.

$CTYPE – Contains the name of a locale that specifies the case conversion conventions to be used.

Enhanced System Functions

$LOWERCASE – Returns an all-lowercase string based on the conventions specified by the locale setting in the $CTYPE system variable.

$UPPERCASE – Returns an all-uppercase string based on the conventions specified by the locale setting in the $CTYPE system variable.

$SEARCH – Searches string-1 for first occurrence of string-2 and returns its character position (or 0 if not found). If called for, a case-insensitive search uses case-conversion rules specified by the locale setting of the $CTYPE system variable.

$TRIM – Removes a character from a designated string. If the character to be removed is alphabetic and case conversion is required, it proceeds according to the rules specified by the locale setting of the $CTYPE system variable.

8.10.1. Sample Code Routines

8.10.1.1. CCS Keyword Parameter (OLD Command)

Format

```
OLD  [  CCS = {  CURRENT | FILE | ccs-id  }  ]
```

Values

CURRENT

transliterates input data to the working CSS (The CCS of the currently visible editing object. If you save a file or element from the workspace, it will be marked with the working CCS). This is the default setting.

FILE

transliterates input data to the CCS setting of the control image first encountered in the data.. This parameter setting also changes $CCS to this value.

ccs-id

transliterates input data to the CCS identified by the CCS ID. A CCS ID may be specified by name or number. This parameter setting also changes $CCS to this value.

8.10.1.2. $LOWERCASE System Function

Format

```
$LOWERCASE (string-expression)
```

where *string-expression* is the character string to be converted to lowercase.

Example

```
$LOWERCASE ("JÄGER")
```

returns uppercase JÄGER as lowercase jäger.

8.10.1.3. $SEARCH System Function

Format

```
$SEARCH(string1,string2 [,start] )
```

Values

string-1

is the main character string.

string-2

is the character string within *string-1*.

start

is an optional integer that indicates the column number where the search should begin. If not specified, column 1 is assumed.

Example 1

```
$SEARCH('understand','n',3)
```

returns 9, the column number for n (*string-2*), which starts at column 3 within understand (*string-1*).

Example 2: For alphabetical characters only

```
$CASESENSITIVE : = ON

$SEARCH (JÄGER,ä)
```

returns zero (0) because ä is not found.

Example 3: For alphabetical characters only

```
$CASESENSITIVE : = OFF

$SEARCH (JÄGER,ä)
```

returns 2 because Äand ä are equivalent.

8.10.1.4. $TRIM System Function

Format

```
$TRIM(string [,side,character] )
```

Values

string

is the string from which alpha characters are to be removed.

side

is the side of the string from which the characters should be removed. Values are either RIGHT (default), LEFT, BOTH, or ALL.

character

is any single character. If not specified, the blank character is assumed.

Example 1: All blank characters are removed.

```
$TRIM("A B C",all)
```

returns ABC (all blank characters removed).

Example 2: Alpha character in specified case is removed.

```
$CASE : = ON
$TRIM (ÄäBC,ALL,ä)
```

returns ÄBC. Only the lowercase ä is removed.

Example 3: Alpha characters in either case are removed according to specified value.

```
$CASE : = OFF
$TRIM (ÄäBC,ALL,ä)
```

returns ÄBC. Only the lowercase ä is removed.

returns BC. Both Ä and ä are removed.

8.10.1.5. $UPPERCASE System Function

Format

```
$UPPERCASE (string-expression)
```

where *string-expression* is the character string to be converted to uppercase.

Example

```
$UPPERCASE ("jäger")
```

returns lowercase jäger as uppercase JÄGER.

8.10.1.6. $CCS System Variable

$CCS allows you to

- Store Interactive Processing Facility data in the CCS of your own language.
- Use a SAVE or REPLACE command to mark SDF files and elements with the working CCS.

Format

```
[SET] $CCS: = "ccs-code"
```

where *ccs-code* is any CCS defined for a valid locale on your system. This is the CCS setting of the workspace if one exists and the I18N Feature is active. If not, $CCS has the value I18N_INACTIVE.

Examples

```
[SET] $CCS : = US_ASCII
```

or

```
[SET] $CCS : = 1
```

Since $CCS may be specified as by either a name or a number, either of these expressions can be used to specify the 7-bit ASCII character set.

```
[SET] $CCS : = "8859-1"
```

The ISO character set 8859-1 (decimal code 35) is the default setting if the I18N Feature is active and no other CCS setting exists.

8.10.1.7. $TERMINALCCS System Variable

$TERMINALCCS (also shown as $TCCS or $TERMCCS) allows you to

- Enter input on and view output from terminals configured with CCSs appropriate to your own language.
- Run the Interactive Processing Facility with different CCS values for $CCS and $TERMINALCCS.

$TERMINALCCS does not apply to

- Interactive Processing Facility commands, keywords, and special characters. These are assumed to be encoded in ASCII.
- File/element input from the OLD command. These are tagged with their own CCSs by way of control images.

Format

```
[SET] $TERMINALCCS: = "ccs-code"
```

where *ccs-code* is any CCS defined for a valid locale on your system.

At the beginning of an Interactive Processing Facility session, *ccs-code* is the CCS setting of the workspace if one exists and the I18N Feature is active. If not, $TERMINALCCS has the value I18N_INACTIVE.

$TERMINALCCS may be specified either by name or by number.

Example

```
[SET]  $TERMINALCCS: = "8859-1"
```

The ISO character set 8859-1 (decimal code 35) is the default setting if the I18N Feature is active and no other CCS setting exists.

8.10.1.8. $COLLATE System Variable

$COLLATE allows you to search, order, and sort characters strings according to the character comparison conventions of your own language.

Format

```
[SET]  $COLLATE: = "locale"
```

where **locale** is any valid locale on your system. This locale controls the rules for collating characters.

At the beginning of an Interactive Processing Facility session, $COLLATE is set to the value of the user's environment variable LC_COLLATE if this variable is defined. If it is not, a system-defined default is used.

Example

```
[SET]  $COLLATE: = "EN_US.ASCII"
```

The locale EN_US.ASCII is the current setting of the run-level environment variable LC_COLLATE if not defined.

8.10.1.9. Sample Procedure Demonstrating $COLLATE

This procedure demonstrates how changing the value of the Interactive Processing Facility system variable COLLATE will change the result of a simple sort program.

You can insert this procedure into a symbolic element of subtype IPF and enter the name of the element to run it. Then you can set up a test file and populate it with element sort-data (test data) and element sort-proc (sorting procedure).

```
@
Create sort-test*file. acce=publ
use pf.,sort-test*file.
new pf.sort-data
1 cerro
2 chorro
3 lobbo
4 lindo
5 llamar
6 cuchillo
7 casa
save
dis "This is the original, unsorted worksplace."
p a
@
@ Create the IPF procedure sort-proc, which uses a simple
@ insertion algorithm to sort the contents of the workspace.
@
new pf.sort-proc;type ipf
1 Do %i from=1 to=$b-1
```

```
2   % :=%i
3   %x := $text(%i)
4   do %j from=%i+1 to=$b
5     if $text(%j) <%x
6        %k := %x := $text(%j)
7     endif
8   enddo
9   generate $text(%i),,%k,,overwrite
10    generate %x,,%i,,overwrite
11  enddo
12  return
save
@
@ Set $COLLATE to en_US.8859-1 and sort the workspace.
@
set $collate:="en_US.8859-1"
dis "The current value of $COLLATE is " & $COLLATE
pf.sort-proc
dis "This is the workspace, sorted according to English-language"
    && "conventions."
p a
@
@ Read unsorted element sort-data back into workspace, set
@ $COLLATE to es_ES.8859-1, and sort the workspace according
@ to Spanish conventions.
@
old pf.sort-data
dis "This is the original, unsorted workspace."
p a
set $collate:="es_ES.8859-1"
dis "The current value of $collate is " & $COLLATE
pf.sort-proc
dis "This is the workspace, sorted according to
    Spanish-language" && "conventions."
p a
@
@ Read unsorted element sort-data back into workspace, set
@ $COLLATE to 13 (same as es_ES.8859-1), and sort the
@ workspace according to Spanish conventions.
@
old pf.sort-data
dis "This is the original, unsorted workspace."
p a
set &COLLATE:=13
dis "The current value of $COLLATE is " & $COLLATE
pf.sort-proc
dis "This is the workspace, sorted according to
@ Spanish-language" && "conventions."
p a
@
@ Do cleanup.
```

```
@
purge sort-test*file.
```

8.10.1.10. $CTYPE System Variable

$CTYPE allows you to perform case-sensitive operations using the case conversion conventions of your own language.

Format

```
[SET]  $CTYPE: = "locale"
```

where *locale* is any valid locale on your system. This locale controls the rules for character classification.

At the beginning of an Interactive Processing Facility session, $CTYPE is set to the value of the user's environment variable LC_CTYPE if this variable is defined. If it is not, a system-defined default is used.

Example

```
[SET]  $CTYPE: = "en_US.ASCII"
```

where en_US.ASCII is the current setting of the run-level environment variable LC_TYPE if not defined.

8.10.1.11. Affect of $CTYPE and $COLLATE on Procedure Statements

Affect on $CASE Statement

If the value of the Interactive Processing Facility system variable $CASE is set to FALSE, the ASCII -based mechanism for lowercase conversion is replaced with a call to I18NLIB using the locale setting of $CTYPE.

If the $CASE setting defines a character classification theme that is ASCII-like, no user-visible change takes place.

Once any needed lowercase conversion has occurred, another call is made to I18NLIB, this time using the locale setting of $COLLATE, to handle the actual character comparisons.

Affect on IF, REPEAT, and WHILE Statements

When two strings are compared in a conditional expression and $CASE is FALSE, the conversion mechanism uses an I18NLIB service routine controlled by the locale setting of $CTYPE.

For string comparisons, the Interactive Processing Facility comparison mechanism is replaced by a call to an I18NLIB service routine controlled by the locale setting of $COLLATE.

8.10.2. Configuring, Activating, and Deactivating the I18N Feature

Procedure

1. Execute the COMUS CONFIGURE command to display the Main Menu for IPF CONFIGURE.
2. Select option 2 for the Internationalization (I18N) feature parameters.
3. Press transmit to display the I18N FEATURE screen containing the I18N_ACTIVE parameter.
   - When the I18N Feature is activated, the Interactive Processing Facility handles text using the conventions of the user's natural language. This includes the ability to
     – Handle different coded character sets (CCS).
     – Transliterate text from one character set to another.
   - When the I18N Feature is deactivated, the Interactive Processing Facility handles text using the rules of the ASCII character set.

8.10.3. Calling Procedures with I18N Feature Active

Execution Restricted to a Single CCS

The execution of an Interactive Processing Facility procedure is restricted to a single coded character set (CCS). The CCS for the first significant image is assumed for the entire procedure.

Conversion to SDF Files Recommended

Data from special-format procedures is treated as being coded in ASCII. While this may work in most cases, it is suggested that you use the CONVERT utility to change the Interactive Processing Facility special-format omnibus file to a symbolic file (SDF).

8.10.4. Error Messages

**IPF Error Messages

Table 8–2. **IPF Error Messages

If you receive this error message	It means	And you need to
**IPF160	Your workspace is in an unrecognized CCS and has been marked as being in the default CCS. The values of $CCS and $TERMINALCCS have been set to their default values.	Do nothing--this is an informational message only.

Table 8–2. **IPF Error Messages

If you receive this error message	It means	And you need to
**IPF170	Your workspace has been recovered successfully after the abnormal termination of your previous session. The values of $CCS and $TERMINALCCS have been set to the CCS value reflected in your workspace file if that value is available and corresponds to a valid CCS on your system. If this value is not available or valid, the values of $CCS and $TERMINALCCS have been reset to their default values.	Do nothing--this is an informational message only.
**IPF180	Your workspace is now empty because it could not be recovered successfully after the abnormal termination of your previous session. The values of $CCS and $TERMINALCCS have been reset to their default values.	Do nothing--this is an informational message only.
**IPF280	Your workspace file has been destroyed and cannot be recovered. A new workspace file has been created. The values of $CCS and $TERMINALCCS have been reset to their default values.	Do nothing--this is an informational message only.
**IPF300	The I18N Feature has been deactivated because an unexpected I18N error status was received.	Contact your system administrator
**IPF420	While processing your IN command, an unexpected error status was received from the OPEN_CCS_TO_CCS service routine.	Contact your system administrator.
**IPF1520	An internal error has been detected in the system function.	Contact your system administrator.
**IPF1740	An error status was reported from the CCS_TO_CCS service routine. Input data is still being processed, but it is no longer being transliterated from the CCS of $TERMINALCCS to the CCS of $CCS.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.
**IPF1750	An image with an invalid CCS was found within the file on your IN command.	Ask your system administrator to register this CCS on your system.
**IPF1760	An error status was reported from the CCS_TO_CCS service routine. Input data from your IN command is still being processed, but it is no longer being transliterated from the CCS of the data image to the CCS of $CCS.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.

Table 8–2. **IPF Error Messages

If you receive this error message	It means	And you need to
**IPF1770	An error status was reported from the CCS_TO_CCS service routine. Output data is still being processed, but it is no longer being transliterated from the CCS of $CCS to the CCS of $TERMINALCCS.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.
**IPF1780	The output data has been truncated because it exceeded the character limit, but it is still being transliterated.	Review the output data to determined what was truncated.
**IPF1995	The process could not be completed, due to an internal error in the Interactive Processing Facility Command Analyzer.	Contact your system administrator.
**IPF5040	Some of the print file data in your lookspace could not be transliterated because one or more images in the file are in a CCS that is not defined on your system.	Use the OLD command to bring a valid print file into your lookspace.
**IPF 6010	An unexpected error status was received from the OPEN_CCS_TO_CCS service routine during reload.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.
**IPF 6020	An unexpected I18N error status caused an Interactive Processing Facility internal error. Minimal processing may continue, but I18N-related operations may not produce the expected results.	Contact your system administrator.
**IPF6210	The $TRIM function was not executed because the uppercase conversion of the third parameter yielded unexpected results. If the system variable $CASESENSITIVE is set to FALSE, uppercase conversion in certain languages may result in two characters.	Set this variable to TRUE (for a case-sensitive search) and enter the command again.
**IPF6530	The $TEXT system function could not be completed, due to an I18N internal error.	Contact your system administrator.

**EDIT Error Messages

Table 8–3. **EDIT Error Messages

If you receive this error message	It means	And you need to
**EDIT2200	The command was not executed because the case conversion of a string yielded unexpected results. If the system variable $CASESENSITIVE is set to FALSE, case conversion in certain languages may expand or contract the string.	Set this variable to TRUE (for a case-sensitive search) and enter the command again.
**EDIT2300	The process could not be completed because the I18NLIB service routine returned an invalid status.	Contact your system administrator.
**EDIT5050	Some or all of the data in a file could not be converted because it uses a CCS that is not defined on your system.	Ask your system administrator to register the CCS on your system.
**EDIT5060	The process could not be completed due to an I18N internal error.	Contact your system administrator.
**EDIT6470	An error status was reported from the CCS_TO_CCS service routine. Input data is still being processed, but it is no longer being transliterated from the CCS of $TERMINALCCS to the CCS of $CCS.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.
**EDIT6480	An error status was reported from the CCS_TO_CCS service routine. Output data is still being processed, but it is no longer being transliterated from the CCS of $CCS to the CCS of $TERMINALCCS.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.
**EDIT6490	The output data has been truncated because it exceeded the character limit, but it is still being transliterated.	Review the output data to determined what was truncated.

**PROC Error Messages

Table 8–4. **PROC Error Messages

If you receive this error message	It means	And you need to
**PROC5300	The CCS of the first significant image within the procedure could not be determined.	Ask your system administrator to register the CCS of the procedure on your system.
**PROC5310	An unexpected error status was reported from the CCS_TO_CCS service routine. The procedure commands are still being processed, but they are no longer being transliterated from the CCS of the first significant data image within the procedure to the CCS of $CCS.	Contact your system administrator.

**SQL Error Messages

Table 8–5. **SQL Error Messages

If you receive this error message	It means	And you need to
**SQL201980	An unexpected error status was reported from the CCS_TO_CCS service routine. Output data is still being processed, but it is no longer being transliterated from the CCS of $CCS to the CCS of $TERMINALCCS.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.

**UA Error Messages

Table 8–6. **UA Error Messages

If you receive this error message	It means	And you need to
**UA570	An unexpected error status was reported from the CCS_TO_CCS service routine. Input data is still being processed, but it is no longer being transliterated from the CCS of $TERMINALCCS to the CCS of $CCS.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.

**VARIABLE Error Messages

Table 8–7. **VARIABLE Error Messages

If you receive this error message	It means	And you need to
**VARIABLE40	The value of this system variable cannot be changed unless the I18N Feature is activated.	Ask your system administrator to activate the I18N Feature on your system.
**VARIABLE50	Your SET or ACCEPT command was rejected. The current editing object is not large enough to permit a change from the working CCS to the CCS you specified.	If you are working in the workspace and you created the workspace file, you may either (1) recreate the file with a larger maximum size or (2) use the EXECUTE command to assign a larger size. For an example, see "Using the EXECUTE Command to Assign a Larger File Size" after this table.
**VARIABLE60	The images could not be converted to the working CCS because they are in a CCS that is not defined on your system.	Ask your system administrator to register the CCS of the images on your system.
**VARIABLE70	One or more images in the current editing object has been truncated because it exceeds the maximum character length.	Check your current editing object for possible truncation of images.
**VARIABLE80	The specified system variable has not been changed.	Contact your system administrator.
**VARIABLE85	The system variable $TERMINALCCS cannot be set to Fieldata.	Specify a different CCS.
**VARIABLE90	The system variable $CCS cannot be set to Fieldata.	Specify a different CCS.
**VARIABLE300	An unexpected error status was reported from the CCS_TO_CCS service routine. Output data is still being processed, but it is no longer being transliterated from the CCS of $CCS to the CCS of $TERMINALCCS.	Contact your system administrator. As an interim solution, set the value of $TERMINALCCS to equal the value of $CCS.
**VARIABLE310	The output data has been truncated because it exceeded the character limit, but it is still being transliterated.	Review the output data to determined what was truncated.

Using the EXECUTE Command to Assign a Larger File Size

```
EXECUTE "@ASG,A  IPF$userid.,///size"
```

where **userid** is your user-id and **size** specifies the maximum allowable length for the workspace file

**WORKFILE Error Messages

Table 8–8. **WORKFILE Error Messages

If you receive this error message	It means	And you need to
**WORKFILE550	The CCS parameter with the OLD command cannot be set to Fieldata.	Specify a different CCS.
**WORKFILE665	The OLD command has been rejected because it causes the active editing objext to expand beyond its maximum size.	If you created the workspace file, recreate it with a larger maximum size. If you want a larger lookspace file, your system administrator will need to reconfigure it.
**WORKFILE720	The OLD command has been rejected because the CCS of the first significant data image within the file could not be determined.	Ask your system administrator to register the file CCS on your system.
**WORKFILE730	The OLD command could not transliterate some or all of the data in the file because the data is in a CCS that is not defined on your system.	Review the file and the active editing object to determine which images are affected. Ask your system administrator to ensure that all CCSs present in the file are defined on your system.
**WORKFILE740	The OLD command completed properly, but the images in the file were converted from Fieldata to US ASCII.	Do nothing–this is an informational message only.
**WORKFILE750	An error status was reported from an I18NLIB service routine while executing the OLD command. The OLD command may or may not have been executed properly. The workspace may have been changed, left unchanged, or reinitialized.	Contact your system administrator.

Table 8–8. **WORKFILE Error Messages

If you receive this error message	It means	And you need to
**WORKFILE780	The OLD command was rejected because an invalid CCS was specified with the CCS keyboard parameter.	Verify that the CCS is valid for your sytem and that the CCS keyboard parameter was typed correctly. Ask your system administrator to register the CCS on your system.
**WORKFILE790	A bad status internal error was returned.	Contact your system administrator.

8.11. IS-6000 Terminal Emulator

Internationalized IS-6000

- Supports sending and receiving of both 7-bit and 8-bit character codes.
- Supports UTS emulation for 7-bit and 8-bit ASYNC terminals.
- Supports certain UNIX applications.

UNIX Applications Using Remote UTS Access Programming Interface

In this environment, IS-6000 reports the terminal's assumed coded character set (CCS) to user applications and allows them to select an alternate CCS (if the terminal supports this capability).

UNIX Applications Using IS Programming Interface

In this environment, IS-6000 reports the CCS of the terminal or the user application.

8.12. Logic and Information Network Compiler (LINC II)

Internationalized LINC II

- Includes multi-language capabilities.
- Provides coded character set identifiers (CCS ID) for I18N terminals.
- Includes system data item GLB.CCS.

Multi-Language Capabilities

- Uses standard COBOL I/O to read and write extract files.
- May use external COBOL routines to read and write SDF files containing CCS-tagged data and make the CCS ID available to a LINC II application.

- Can select many different data formats during program execution in the language chosen by the user.
- Enables translation of terminal screen formats, message text literals, and constant data without modifying application code.

Coded Character Set Identifiers (CCS ID)

- Allows use of multiple character sets within a single LINC II application.
- Recognizes CCS ID passed from user terminal on input from I18N terminal.
- Allows developers to save CCS information for use later.

System Data Item IGLB.CCS

- Used to pass the CCS ID from the user terminal to a LINC II application program.
- Defined as 5-digit numeric item (EDIT;N LENGTH;5)
- Can be used as is or as an index into a table containing more meaningful CCS descriptions.

8.13. Message Control Bank (MCB)

Internationalized MCB

- Supports 7-bit and 8-bit characters. Internationalized MCB can transmit 7-bit and 8-bit characters. However, transaction names cannot contain 8-bit characters.
- Provides terminal handlers for character conversions. Internationalized MCB provides special terminal handlers to convert between 7-bit and 8-bit character representations.
- Includes standard transliteration tables and customizing capabilities.

For information on naming transactions so that they can be transmitted in an I18N environment, refer to the *Transaction Processing Administration and Operations Reference Manual* (7830 7881).

8.13.1. MCB Transliteration Service

The MCB transliteration service helps to migrate a terminal network to 8-bit terminals with minimal impact on existing 7-bit applications. A number of standard conversion tables are provided to assist with transliteration.

8.13.2. General Constraints

MCB-Connected Transactions Only

This MCB transliteration service only occurs with MCB-connected transactions. The COMPOOL interface does not have access to the coded character set, which is required to determine the need for transliteration.

No Use of Write-Protected Banks

Transliteration changes the text area in the data buffers of the transaction. For this reason, the data cannot be passed to the MCB in a write-protected bank.

Default Conversion of ISO 8859 8-Bit Characters

Any ISO 8859 8-bit character that does not have an equivalent ISO 646 character is converted to a question mark (?) on input to the transaction.

Unwanted Character Conversions

The transliteration service tries to bypass all UNISCOPE screen control sequences to avoid changing these characters. Bypassing is not always possible if a control sequence is split across two partial outputs to the MCB. If this sequence contains characters that are subject to conversion by the transliteration routines, the results will be unpredictable.

Note: *To avoid unwanted transliterations, use the standard conversion tables.*

Input Constraints

Input Source Must Use ISO 8859 Character Sets

Input transliteration can occur only if the terminal or emulator that generated the input is using one of the ISO 8859 character sets.

Transactions Must Read MPA

Input transliteration cannot occur if transactions using the primitive interface do not read the MPA.

Output Constraints

Output Must be to Same PID

Output transliteration can occur only if transliteration is required on input, and the output is to the same PID. If the output is sent to a different PID, the output transliteration procedures are not called.

Output Cannot be Unsolicited

Output transliteration cannot occur when the output is unsolicited.

Transactions Must Read MPA

Output transliteration cannot occur if transactions using the primitive interface do not read the MPA.

Passoff Limitations

Output transliteration cannot occur when the transaction performs a passoff to another transaction.

Exceptions to Passoff Limitations

- Output transliteration can occur with passoffs if a transaction can send output to the originating PID immediately after a passoff to another transaction.
- Output transliteration can occur on output to a PID from a passoff transaction, if the passoff message contains the same contents of the original AUX area, and specifies the same PID.

8.13.3. List of Element Names in I18N$*UTILITY File

In the standard MCB transliteration tables, these names are used for the symbolic and relocatable elements in the file I18N$*UTILITY.

This element	is for use in
CONTABUS	United States
CONTABUK	United Kingdom
CONTABSWE	Sweden
CONTABSWI	Switzerland
CONTABFIN	Finland
CONTABNL	The Netherlands
CONTABFRA	France
CONTABNOR	Norway and Denmark
CONTABGER	Germany
CONTABITA	Italy
CONTABESP	Spain

Example: Mapping for Multiple Conversions

If the same transaction is required to handle German, Italian, and French, you can map it to be called with three different transaction codes. Each code identifies a separately mapped version of the transaction–one version for each language. The appropriate conversion table is included.

8.13.4. @MAP Directives

The @MAP source for your transaction should include the following directives:

```
LIB   I18N$*UTILITY      . to get the new relocatables.
LIB   MCBn*MCB$          . to satisfy the MCB entry points.

                         . MCBn*MCB$ is the file installed
                         . by the SOLAR INSTALL command
                         . the specified application.

LIB   TIP$*TIPLIB$          . to get those relocatables not
                            .  supplied in I18N$*UTILITY.
.
.
IN   I18N$*UTILITY.CONTABxxx

                         . This is one of the conversion tables
                         . as supplied or a locally generated
                         . equivalent. It is a read-only element
```

```
                        . so it may be included in the IBANK
                        . that contains the MCB interface
                        . relocatables.

IN   I18N$*UTILITY.INPUT8859
IN   I18N$*UTILITY.OUTPUT885

                        . These two IN statements are optional.
                        . The elements will be included in the
                        . collection, as directed by the LIB
                        . statement for I18N$*UTILITY.  The IN
                        . statements allow you to place them in
                        . a specific order.  They must be in the
                        . same bank as the CONTABxxx element.
```

8.13.5. Sample Code for User Call

Collecting a Transaction for MCB Transliteration

The MCB transliteration code can be called directly from user programs. MASM code sequences are required for transliteration of input and output.

Example 1: Sample Code for Input Transliteration

```
L   R1,byte-count        . byte count of text
L   A0,start-address     . start word address of text
L   A5,offset            . offset into first word
                         . 0 = Q1
                         . 1 = Q2
                         . 2 = Q3
                         . 3 = Q4
LMJ   X11,I18NINP        . enter the code
```

In addition to the above registers, the primitives destroy the contents of A1 through A4.

Example 2: Sample Code for Output Transliteration

```
L   R1,byte-count        . byte count of text
L   A2,start-address     . start word address of text
L   A1,offset            . offset into first word
                         . 0 = Q1
                         . 1 = Q2
                         . 2 = Q3
                         . 3 = Q4
LMJ   X11,I18NOUT        . enter the code
```

In addition to the above registers, the primitives destroy the contents of A0, A3, and A4.

Additional Elements to Include in the Input/Output Code Sequences

When collecting your user program, you also need to include these three additional elements, which are contained in the I18N$*UTILITY file: INPUT8859, OUTPUT8859,

and CONTAB. One of the CONTAB elements is required in order to provide the conversion table.

8.14. Network Database Server

Note: *The Network Database Server was formerly known as DMS 2200, which remains the installation and support name.*

Internationalized Network Database Server

- Supports sets that are ordered according to regional and national sorting schemes.
- Sorting is based on a specified locale.
- Locale is identified by the LOCALE clause on the SCHEMA statement in the schema definition.

8.14.1. LOCALE Clause in SCHEMA Statement

Network Database Server processing is turned ON by using the LOCALE clause on the SCHEMA statement.

Specifying a Locale

Use the optional I18N syntax in the schema to specify the locale for the schema. This syntax appears immediately after the SCHEMA statement.

```
[SCHEMA LOCALE IS {SYSTEM DEFAULT / RUN DEFAULT / NONE / locale-name} ]
```

where **locale-name** is a valid name for a locale that is available on your system.

Results of Not Specifying a Locale

If you specify **locale NONE** or the LOCALE clause is absent, no calls are made to the I18N Service Library, and the ASCII (binary) collating sequence is used.

8.14.2. Sorted Chain Sets

I18N Schema Changes

Internationalizing the Network Database Server has required the following schema changes, so that the DMR can use a locale when collating the keys of a sorted chain set.

- The LOCALE clause must be specified in the SCHEMA statement.
- The sort key items must have a usage of DISP-I18N.
- The I18N DDL and DMR must be used.
- Using the I18N COBOL compiles are recommended. The I18N ADMLP will work, but it will give warnings on the DISP-I18N clause.

8.14.3. FIND/FETCH Command

If the keys for the sorted chain sets have been internationalized, no additional specification is necessary to make FETCH formats 6 and 7 function properly in honoring the collation sequence of the specified locale.

8.14.4. Index Sequential Record Placement

Internationalizing the Network Database Server has required the following schema changes, so that the DMR can use a locale when collating the keys of records with location mode index-sequential.

- A LOCALE clause must be specified in the SCHEMA statement.
- For any given record, if the first sort key item has a usage of DISP-I18N, then all other sort key items are assumed to have usage of DISP-I18N, regardless of their actual usage.
- I18N DDL and DMR must be used.

Using the I18N COBOL compiler are recommended. The I18N ADMLP will work, but it will give warnings on the DISP-I18N clause.

8.14.5. Schema Absolute and DMR Levels

Modifying a Schema and a DMR Independently

You can modify a schema and the current level of a DMR independently, but the results may be unpredictable.

Possible Error Situations for Runs Using Earlier Versions of a DMR

Depending on what the program attempts to do, the following errors could occur:

- The run will execute as if the I18NLIB were not in use.
- Errors dealing with sorting order may appear.
- Previously existing records that are loaded with the I18N DMR might not be found.

Possible Scenarios With Runs Using a New DMR

No errors occur with initial load programs, but the following scenarios are possible for later runs using a new DMR:

If the DMR is	And the schema accessed by the application is	The following result occurs
A level with I18N features	An earlier version without I18N features	The schema continues to operate; it does not need reprocessing.
Reverted to a level without I18N features	Created with a new Data Definition Language (DDL) with I18N features OFF	The application executes as usual.
Reverted to a level without I18N features	Created with a new DDL with I18N features ON	The application receives a schema abort at IMPART time.

8.14.6. C Compiler Interface

The Network Database Server cannot be accessed directly from the C compiler.

Interlanguage Call Required

To access the Network Database Server from the C compiler, make an interlanguage call to either the COBOL compiler or the FORTRAN.

8.14.7. COBOL Compiler Interfaces

The COBOL compiler interfaces to the Network Database Server use the data management communications area (DMCA).

IO-ERR-STATUS Field Renamed

In the COBOL environment, the RENAMED clause is used to rename this field the SECONDARY-ERR-STATUS field. This renamed field contains the error code returned by other modules called from the DMR.

Placement of Error Codes

- If you are using the COBOL compiler, error codes returned from I18NLIB are placed in the *SECONDARY-ERR-STATUS* field.
- If you are using ADMLP, error codes returned from I18NLIB are placed in the *IO-ERR-STATUS* field.

8.14.8. FORTRAN Compiler Interface

The FORTRAN Compiler interface to the Network Database Server uses the data management communications area (DMCA).

D$IOER Field Renamed

In the FORTRAN environment, this field is renamed the D$SERS field. D$SERS contains the error code returned by other modules called from the DMR.

Placement of Error Codes

- If you are using the FORTRAN Compiler, error codes returned from I18NLIB are placed in the *D$SERS* field.
- If you are using FDMLP, error codes returned from I8NLIB are placed in the *D$IOER* field.

8.15. ODBC Access

Note: *ODBC Access incorporates the products formerly known as INFOAccess and UniAccess ODBC.*

Internationalized ODBC Access

- Includes NCHAR data type.
- Supports conversion between wide and multibyte characters.

- Translates NCHAR data to either Shift-JIS or LETS-J
- Converts Shift-JIS/LETS-J to the encoding scheme of the Network Database Server (formerly DMS 2200).
- Uses CHAR data type to get CHAR/NCHAR data from the Relational Database Server (formerly RDMS 2200).

Addition of NCHAR Data Type

On the OS 2200 server side, an NCHAR data type has been added to the ODBC Access interface to permit use of 16-bit character sets. This data type is used to convey the DDL Display-2 data to the client side.

Support for Conversion Between Wide and Multibyte Characters

The NCHAR data type is based on the WCHAR_T type of the C programming language. The C standard library offers limited support for conversion between these wide characters and multibyte characters. Characters are ordered and compared based on their binary values, but there is no support for uppercasing, lowercasing, or conversions between NCHAR columns and numeric values.

Translating NCHAR Data and Shift-JIS/LETS-J

The standard ODBC function calls SQL Data Source To Driver and SQL Driver to Data Source are used for both translations. The driver assumes that the ODBC SQL SET Connect Option function call is always made by the client application to request a translation.

Translating NCHAR Data

The client ODBC driver translates all NCHAR data coming from network layers to Shift-JIS (PC client) or LETS-J (2200 client). This includes incoming NCHAR data from the ODBC interface on a ClearPath OS 2200 server.

Converting Shift-JIS/LETS-J

The outgoing Shift-JIS or LETS-J is converted to the encoding scheme of the Network Database Server immediately after being submitted to network layers. Any application requesting NCHAR data can make sense of Shift-JIS/LETS-J and the proprietary coding scheme of the Network Database Server.

Use of CHAR Data Type

OBDC uses the CHAR data type to get information from both CHAR and NCHAR type columns in the Relational Database Server. An application has to know which columns are CHAR and which are NCHAR. All data from the NCHAR column is translated between Shift-JIS and LETS-J.

Note: *Due to a Microsoft constraint, the Relational Database Server cannot use the ODBC interface to inform an application that a column is NCHAR.*

8.16. Oracle Database Access

Note: *This topic deals with access to the Oracle database as it applies to the Relational Database Server. This access was formerly referred to as OTG for RDMS, which remains the installation and support name.*

Internationalized Access to the Oracle Database

- Supports National Language capabilities of Oracle.
- Enables client applications to manipulate server data in character sets of their choice.
- Includes two new function calls that support the NCHAR data type for the Relational Database Server.

Choice of Character Sets

Client applications working in one coded character set (CCS) may present SGL commands to the Relational Database Server by way of an Oracle7 integrating server that could be configured in another possibly different CCS.

Assumptions and Possible Limitations

- Assumptions
  - The gateway is considered to be running in the character set specified as an initialization parameter.
  - Client applications and Oracle7 may be running in a different character set.
- Limitations
  - There may be linguistic differences when post-processing occurs in Oracle7 for commands that the Relational Database Server cannot handle.
  - The replacement character for non-translatable characters may have to be the Oracle7 replacement character, currently a question mark (?).

Use of New Function Calls

The user-written function calls *tg_charset_in*() and *tg_charset_out*() are used to perform character set conversion routines that can be used as follows:

- To replace the gateway stubs by the same name that are shipped with the product.
- To be linked in with the gateway code and be invoked to perform SQL command translation between column data in the character sets of the client tier and /or middleware and that of the desired data store character set.
- To be linked in with the gateway on site and into a product locally.

Sample Code for Function Call tg_charset_in

Function

SGL text received by the gateway from Oracle7 client tools is passed to this routine. The routine converts literals in the text from the gateway character set to the character set for the Relational Database Server.

Syntax

```
unsigned int tg_charset_in ( int       sqlid,
                             char    **sqlstring,
                             int      *sqllength,
                             char     *language,
                             int      *langlength,
                             char    **messagetext)
```

Input Parameters for Function Call tg_charset_in

sqlid	Unique identification of SQL command within user session.
sqlstring	Pointer to SQL command string. This pointer is not necessarily NULL terminated. It may contain national character literals prefixed with N. N'DB1DB2DB3'
sqllength	Length in bytes of SQL command string.
language	Character set identifier of the gateway. This identifier is not necessarily NULL terminated. It uses the Oracle string format with linguistic information included. <language><territory>.<character-set> JAPANESE_JAPAN.JA16EUC JAPANESE_JAPAN.JA16SJIS The character set is the value from which the translation is to be performed.
langlength	Length in bytes of the identification string for the gateway character set.

Output Parameters for Function Call tg_charset_in

qlstring	Pointer to SQL command string. This pointer is not necessarily NULL terminated. It may contain national character literals prefixed with N.N'DB1DB2DB3'
sqllength	Length in bytes of SQL command string. This parameter may increase, due to *realloc*(), if more storage is needed for this string.

messagetext Message text string that identifies a possible conversion error or warning."CNV-<3-digit-number> : <message-text>This string must be 80 characters or less and be NULL terminated. The buffer space is supplied by the caller.

Return Values for User-Written Functions

- Considerations

  The user-written functions *tg_charset_in*() and *tg_charset_out*() may accept or reject the SQL command string.

- Constraints

  If no conversion codes are available for a specified character translation, the string could be accepted with replacement characters for the non-existent codes.

- Possible Conversion Values

If this value appears		It means that
Code Name	**Code Number**	
CONV_OK	0	The conversion/translation succeeded.
CONV_WARN	1	The conversion/translation erred in part, but the command was still able to execute.
CONV_ERROR	2	The conversion/translation failed. The converted string is not meaningful.

8.17. Procedure Definition Processor (PDP)

Internationalized PDP

The I18N version of PDP includes a processor call option to differentiate between kanji (Japanese) characters and standard 8-bit characters.

Handling Embedded Kanji Strings

Without the kanji option, the octal values 0223 and 0224 are interpreted as standard 8-bit characters.

With the kanji option, the following occurs:

This octal value	is interpreted as a	to indicate
0223	shift-in character	the beginning of an embedded kanji string.
0224	shift-out character	the end of an embedded kanji string.

8.18. Processor Common Input/Output System (PCIOS)

Internationalized PCIOS

- Supports 8-bit data storage and retrieval.
- Includes enhanced SSDF and MSAM file functions.

When an application uses PCIOS directly with its own processor interface module (PIM), it can use any CCS ID values.

Support for 8-Bit Data Storage and Retrieval

PCIOS supports 8-bit data storage and retrieval directly. No special processing is necessary.

Enhanced SSDF and MSAM File Functions

PCIOS allows you to (1) identify the coded character set (CCS) used to represent the data in an SSDF file, and (2) specify the sorting sequence used for the keys in an MSAM file.

Note: *These capabilities are provided for the COBOL Compiler only.*

Creating an Indexed Sequential File Used by PCIOS (COBOL Compiler)

8.18.1. Identifying CCSs in SSDF Files

Process

- For a new SSDF file, the COBOL Compiler stores the value of the variable specified in the CODE TYPE clause within the SSDF file header during the execution of the OPEN OUTPUT statement.
- For an existing SSDF file, the COBOL Compiler stores the value of the CCS identifier within the variable specified in the CODE TYPE clause of the file for which the OPEN INPUT statement is being performed.

Related Topic

CODE TYPE clause on SELECT statement

8.18.2. Specifying Sorting Sequence for Keys in MSAM Files

Procedure

For binary sorting order, PCIOS supports binary collation of 8-bit data directly. No special processing is necessary.

For regional or national sorting order, follow these steps:

1. Create two key fields, one with the displayable text in it and one with an ordering key based on the displayable text.
2. Use the STRING_TRANSFORM service routine to create the ordering key.

3. Include the name of the locale on the OPEN FILE statement. The COBOL Compiler changes the locale name into its corresponding locale number and writes the locale number into the file header record.

8.19. Programmer's Workbench

Note: *The Programmer's Workbench was formerly known as OADE, which remains the installation and support name.*

Internationalized Programmer's Workbench

- Supports kanji (Japanese) characters as well as ASCII characters.
- Includes a context-sensitive help system.

Support for Kanji Characters

Support for kanji characters is activated by NUL as needed.

Setting Parameter for Japanese Character Set

If you are using OADE with Shift-JIS characters in source files or LETS-J on a ClearPath OS 2000 server, you need to set the Japanese Set parameter in the OADE.INI file. No further I18N processing is required.

8.20. Relational Database Server

Note: *The Relational Database Server was formerly known as RDMS 2200, which remains the installation and support name.*

Internationalized Relational Database Server

- Uses seven different interfaces to support column processing for I18N.
- Includes enhanced general processing features.
- Includes new and enhanced features and utilities for schema definition, insert/update processing, and select processing.

Installation Requirement When Collation Order Changes in I18NLIB Levels

Due to updates and enhancements in various I18NLIB levels, the collation order for many locales may differ from the order in earlier levels. If a change has occurred between the current level of I18NLIB and the new level, any area in the Relational Database Server that contains an I18N ordering key based on the locales that changed must be reloaded immediately after installing I18NLIB. For more details on the effects of this change, see 'Relational Database Server: Impact of Locale Collation Differences on I18N Columns.'

Impact of Locale Collation Differences on I18N Columns

Background

Any column in the Relational Database Server that includes the CHARACTER SET and COLLATION clauses is said to be an I18N column. The COLLATION clause contains the name of an I18NLIB locale, so any such column that uses one of the changed locales is impacted.

Requirement

The table containing this kind of column must be reloaded using UNLOAD FORMAT EXTERNAL immediately after installing the I18NLIB so the data is correctly stored and data integrity is maintained. For details, refer to Appendix E of the *Relational Database Server Administration Guide*.

Support for I18N Column Processing

The internationalized Relational Database Server supports I18N processing for columns. This includes storage, sorting, selecting, and ordering of character strings from columns according to the regional or national sorting conventions specified in a locale.

This capability is available through the following Relational Database Server interfaces: IPFSQL; MAPPER; MRI; ODBC; AIS; and the C and COBOL Compilers.

General Processing Features

- Conformance to the SQL 92 standard for I18N processing (see Related Document)
- Use of the I18NLIB service routines.
- Use of the same processing model as the C Compiler.
- Close integration with the I18N features of the COBOL Compiler.

Schema Definition Features

- A character set identifier can be attached to a column.
- A collation identifier can be attached to a column.
- Primary key columns can have CHARACTER SET and COLLATION attributes.
- A secondary index can be created based on a column with CHARACTER SET or COLLATION attributes.

Insert/Update Processing Features

- Ability to verify that the character set of literal values inserted into a column matches the column definition.
- Ability to verify that values moved from one column to another (INSERT SELECT) have comparable character sets.

Select Processing Features

- Support for comparisons in a WHERE clause. This operation is governed by the collation rules that are indicated by the collation identifier in the column definition.
- Support for the inclusion of a specific collation identifier on the ORDER BY clause. This identifier can be different from the collation identifier in the column definition.

Utilities Feature

The LOAD and UNLOAD utilities allow I18N values.

8.20.1. Specifying a Character Set on a Column Definition

Optional CHARACTER SET Clause

The CHARACTER data type specification includes an optional CHARACTER SET clause.

Format

```
CHARACTER  [ (size) ]  [ CHARACTER SET character-set-specification  ]
```

Values

The *character-set-specification* identifies the coded character set (CCS) for the values in the column.

You may specify either the name of (1) a standard character repertoire or (2) an implementation-defined character repertoire. Character repertoire names are not case-sensitive in the Relational Database Server.

A character repertoire name can be that of any character set supported by I18NLIB.

- Standard character repertoire names include ASCII and ISO 8859.1, the most heavily used character sets.
- Implementation-defined character repertoire names include SQL_TEXT, RDMS_TEXT, SQL_CHARACTER, ASCII_GRAPHIC, LATIN1, and ASCII_FULL.
- If you do not specify a character set, the default is RDMS_TEXT.

Examples

```
CHARACTER (12)   CHARACTER SET LATIN1
CHAR (9)  CHARACTER SET SQL_TEXT
CREATE TABLE t1 ( col1 CHARACTER (40)   CHARACTER SET "ISO8859-1" )
```

Rules and Guidelines for Character Sets

Rule 1: A character set specification name must be a legal value.

To be allowable at execution time for the Relational Database Server, a character set specification name must be a legal value for the CCS_NAME_TO_CCS_NUM service routine.

Rule 2: The character set definitions are determined by the I18N Service Library.

The character set definitions must be installed using the I18NLIB character set installation utilities. The I18NLIB character set definition mechanism is the controlling mechanism.

Note: *The appearance of a name in these lists does not guarantee its acceptability at execution time.*

8.20.2. Specifying Collation on a Column Definition

Optional COLLATE Clause

A column definition can include an optional COLLATE clause. The COLLATE clause identifies the rules to use for inserting character values into a column and to use for retrieving values from a column.

Format

```
COLLATE  collation-name
```

Values

The *collation-name* can be shown as either *character-representation* or ".*character-representation*.", which is the name of the locale that describes the desired collation. (For the names of the locales installed on your system, consult your system administrator.)

When the name contains an underscore or period, it must be enclosed in double quotation marks (" "), because collation names are case-sensitive. If a name is not enclosed in quotation marks, the Relational Database Server treats it as if all characters are uppercase.

Examples

```
COLLATE  "en_US.ASCII"

CREATE TABLE t1 ( name CHARACTER (60) CHARACTER SET ISO8859-1 COLLATE
                  C )

CREATE TABLE t2 ( i18nname CHAR(40) CHARACTER SET ISO8859-1 COLLATE
                  "it_IT.8859-1" )
```

8.20.3. Predicate Handling (WHERE Clause)

I18N Impact on Character Columns in Expressions

I18N processing changes the way character columns can be used in expressions. This processing applies to any expression with a comparison operator: <, <=, =, <>, >=, >, BETWEEN, IN, NOT NULL, and EXISTS.

Processing Character Data

The way the Relational Database Server processes character string literals and columns of data type CHARACTER depends on the following conditions:

- If no CHARACTER SET clause was included in the column definition, the column is said to have an implicit character repertoire name.
- If no COLLATE clause was included in the column definition, the column is said to have an implicit collation.

8.20.4. Handling of Implicit and Explicit CHARACTER SET and COLLATE Clauses

Implicit Character Set (Implicit Collation)

Use of Character Repertoire Name RDMS_TEXT

Before I18N features are introduced, all data entered into Relational Database Server databases has the explicit character repertoire name of RDMS_TEXT. This name is assigned if a column definition contains has no explicit character repertoire name; for example

```
CREATE TABLE t1
      (col1 CHAR(20))
```

Exception to Assigning RDMS_TEXT

The existing data in a column may actually be encoded in any character repertoire. The following example assumes that the data actually in the column uses a LATIN1 or ISO 8859.1 character repertoire. If the data in the column actually used an ISO 646 national variant, the results may differ.

Example of Implicit Character Set/Implicit Collation

Implied Ordering

The implied ordering for the character repertoire name follows the binary codes for the character representations. Generally, this means

- All numeric characters collate first, followed by uppercase alphabetic characters, then lowercase alphabetic characters.
- Special characters such as !@#$%^&*() collate before the numerics, between the uppercase and lowercase characters, and after the lowercase characters.
- Characters with diacritical marks generally follow after all base characters (plain letters with no diacritical marks).

Sample Code

Suppose columns *col1* and *col2* are defined as CHARACTER with no explicit character repertoire name and no explicit collation. If a data item of column *col1* has the character value 'X' and a data item of column *col2* is NULL, the values of the following expressions are

```
col1 = 'x',  false  (character values are case-sensitive)
col1 = 'X',  true
col1 < 'r',  true
col1 < 'R',  false
col1 < 'z',  true
col1 < 'Z',  true
col1 < 'Å',  true
col1 BETWEEN 't' AND 'Å',  false
col1 = col2,  unknown
'abc' > col2,  unknown
```

This CREATE TABLE statement

```
CREATE TABLE t1
      (cname CHAR(20))
```

is equivalent to this CREATE TABLE statement.

```
CREATE TABLE t1
      (cname CHAR(20) CHARACTER SET RDMS_TEXT)
```

Explicit Character Set (Implicit Collation)

Assigning Character Repertoire Name

If a column definition has an explicit character repertoire name, the Relational Database Server assigns the specified character repertoire name to the column; for example

```
CREATE TABLE t1
      (col1 CHAR(20) CHARACTER SET "ISO8859-1")
```

When a column is defined with an explicit character repertoire name (other than RDMS_TEXT), the Relational Database Server enforces the use of that name when encoding the values inserted into the column. For example, if column *col1* is as defined above, in the expression

```
INSERT INTO t1 (col1) SELECT col2 FROM t2
```

the column *col2* must also be defined with the clause

```
CHARACTER SET "ISO8859-1"
```

Exceptions for RDMS_TEXT

Columns defined implicitly or explicitly with the character repertoire name RDMS_TEXT can be the source or destination for columns defined with any character repertoire name. In the preceding example, *col2* could also be defined with the character repertoire name RDMS_TEXT.

Matching of Character Repertoire Names

The Relational Database Server also enforces the match of character repertoire names for ESQL variables when the caller is the COBOL Compiler.

Explicit Collation (Implicit Character Set)

The following constraints apply to the internationalized Relational Database Server:

- It does not support attachment of an explicit collation to an implicit character repertoire (RDMS_TEXT).
- It cannot attach a collation without knowledge of the underlying character repertoire used in the column. In this case, the actual underlying character repertoire is known only to the user application.

Explicit Collation (Explicit Character Set)

Defining Columns with Explicit Collations

Columns can be defined with an explicit collation. For example,

```
CREATE TABLE t1 cname CHAR(20) CHARACTER SET "ISO8859-1"
      COLLATE "no_NO.8859-1")
```

Assigning and Matching Character Repertoire Names

The same rules apply as with Explicit Character Set (Implicit Collation)

Using I18NLIB Service Routines

When processing values are in the column, the Relational Database Server uses the I18NLIB service routines with the rules in the specified collation.

8.20.5. Using COLLATE Clause on DECLARE CURSOR ORDER BY Statement

Changing Presentation of Data

Using the COLLATE clause on the ORDER BY statement allows data to be presented differently from how it is stored in the database.

Format

```
DECLARE cursor-name CURSOR FOR query-expression
     [ ORDER BY sort-specification-list
          [ COLLATE collation-name ]  [ ASCENDING | DESCENDING ] ]
```

Values

cursor-name

is the user name for the cursor.

query expression

the expression to be queried, such as

SELECT (no,maxprice FROM clients)

sort-specification-list

can be either the column name, the title name, or an unsigned integer.

collation-name

can be shown as character-representation or " character-representation ", which is the name of the locale that describes the desired collation. (For the names of the locales installed on your system, consult your system administrator.)

When this name contains an underscore or period, it must be enclosed in double quotation marks (" "), because collation names are case-sensitive. If a name is not enclosed in quotation marks, the Relational Database Server treats it as if all characters are uppercase.

Examples

```
DECLARE curs1 CURSOR FOR SELECT ...  ORDER BY col1 COLLATE
"en_US.ASCII"
DECLARE curs2 CURSOR FOR SELECT ...  ORDER BY col_name  COLLATE  C
DECLARE curs3 CURSOR FOR SELECT ...  ORDER BY sort_col COLLATE
"it_IT.8859-1"
```

Rules and Guidelines for Collation Names

Rule 1: A collation name must be a legal value.

To be allowable at Relational Database Server definition or execution time, a collation name must be a legal value for the LOCALE_NAME_TO_LOCALE_NUM service routine. The locale definitions must be installed using the I18NLIB locale installation utilities. The I18NLIB locales installed on your system determine the legal values.

Rule 2: Each collation has an associated character set.

Each collation has an implied or explicit character set associated with it. This association is defined by I18NLIB and can be determined by reports from the Repository for ClearPath OS 2200 (formerly UREP).

The character set must match the explicit or implicit character set of the related Relational Database Server object, as illustrated in the following examples:

- Example 1: CHARACTER SET COLLATE

  If a table contains a column defined with CHARACTER SET ABC COLLATE collation1, then collation1 must be associated with character set ABC.

- Example 2: CHARACTER SET/CURSOR with ORDER BY COLLATE

  If a table contains a column sort_name defined with CHARACTER SET ABC and a CURSOR with ORDER BY sort_name COLLATE collation1, then collation1 must be associated with character set ABC.

8.20.6. Collation Rules for German

The collation rules for the German language in Germany are indicated by "de_DE.8859-1".

General Collating Order

1. Control characters such as STX, ETX, VT, and HT
2. SPACE character
3. Special characters such as !@#$%^&*()-+=
4. Numeric characters 0 through 9
5. Alphabetic characters

Additional Collating Rules for Alphabetic Characters

- Alphabetic characters collate with uppercase and lowercase characters intermixed.
- Uppercase characters precede lowercase characters; for example,

  ```
  A a B b C c D d...
  ```

- Characters with diacritical marks are usually classified with their corresponding base character; for example,

  ```
  "ä" collates with "a"
  ```

- In the German collating scheme, the character "ß" collates as if it were "ss". Thus, the following words compare as equal:

  ```
  "Straße" and "Strasse"
  ```

Sample Code

Suppose columns *col1* and *col2* are defined as having the ISO8859-1 character repertoire name and "de_DE.8859-1" collation.

If a data item of column *col1* has the character value 'X' and a data item of column *col2* is NULL, the values of the following expressions are

```
col1 = 'x',  false  (comparison still case-sensitive)
col1 = 'X',  true
col1 < 'r',  false
col1 < 'R',  false
col1 < 'z',  true
col1 < 'Z',  true
col1 < 'Å',  false
col1 > 'ß',  true
col1 BETWEEN 't' AND 'Å',  false
col1 = col2,  unknown
'abc' > col2, unknown
```

8.20.7. Collation Rules for Norwegian

The collation rules for the Norwegian language in Norway are indicated by "no_NO.8859-1".

General Collating Order

1. Colon (:) and semicolon (;) characters
2. Control characters such as STX, ETX, VT, and HT
3. SPACE character
4. Special characters such as !@#$%^&*()-+=
5. Numeric characters 0 through 9
6. Alphabetic characters

Additional Collating Rules for Alphabetic Characters

- Alphabetic characters collate with uppercase and lowercase characters intermixed.
- Uppercase characters precede lowercase characters; for example,

  ```
  A a B b C c D d...
  ```

- Characters with diacritical marks are usually classified with their corresponding base character; for example,

  ```
  "ã" collates with "a"
  ```

- In the Norwegian collating scheme, the following characters collate after "z" in this order:

  ```
  Æ Ä æ ä Ø Ö ø ö Å å
  ```

Sample Code

Suppose columns **col1** and **col2** are defined as having the ISO 8859-1 character repertoire name and the "no_NO.8859-1" collation.

If a data item of column **col1** has the character value 'X' and a data item of column **col2** is NULL, the values of the following expressions are

```
col1 = 'x',  false  (comparison still sensitive to
                     upper/lowercasing)
col1 = 'X',  true
col1 < 'r',  false
col1 < 'R',  false
col1 < 'z',  true
col1 < 'Z',  true
col1 < 'Å',  true
col1 BETWEEN 't' AND 'Å',  true
col1 = col2,  unknown
'abc' > col2,  unknown
```

8.20.8. Collation Rules for Spanish

The collation rules for the Spanish language in Spain are indicated by "es_ES.8859-1".

General Collating Order

1. Control characters such as STX, ETX, VT, and HT
2. SPACE character
3. Special characters such as !@#$%^&*()-+=
4. Numeric characters 0 through 9
5. Alphabetic characters

Additional Collating Rules for Alphabetic Characters

- Alphabetic characters collate with uppercase and lowercase characters intermixed.
- Uppercase characters precede lowercase characters; for example,

  ```
  A a B b C c D d...
  ```

- Characters with diacritical marks are usually classified with their corresponding base character; for example,

  ```
  "ã" collates with "a"
  ```

- In the Spanish collating scheme, the characters "CH" and "ch" collate between "cz" and "da". Thus, the correct ordering of the words

  ```
  "chorizo"
  "czar",
  "day"
  ```

  according to the Spanish collating scheme is

  ```
  "czar"
  "chorizo"
  "day"
  ```

- Similarly, the characters "LL" and "ll" collate between "lz" and "ma". Thus, the correct ordering of the words

  ```
  "llama"
  "lyric"
  "man"
  ```

  according to the Spanish collating scheme is

  ```
  "lyric"
  "llama"
  "man"
  ```

Sample Code

Suppose columns *col1* and *col2* are defined as having the ISO 8859-1 character repertoire name and "es_ES.8859-1" collation.

If a data item of column *col1* has the character value 'X' and a data item of column *col2* is NULL, the values of the following expressions are

```
col1 = 'x',  false  (comparison still case-sensitive)
col1 = 'X',  true
col1 < 'r',  false
col1 < 'R',  false
col1 < 'z',  true
col1 < 'Z',  true
col1 < 'Å',  false
col1 > 'ß',  true
```

```
col1 BETWEEN 't' AND 'Å',  false
col1 = col2,  unknown
'abc' > col2,  unknown
```

If a data item of column col1 has the character value 'Chorizo', the values of the following expressions are

```
col1 = 'chorizo',  false  (comparison still case-sensitive)
col1 = 'Chorizo',  true
col1 < 'cz',  false
col1 < 'CZ',  false
col1 < 'd',  true
col1 BETWEEN 'czar' AND 'day',  true
```

8.21. Repository for ClearPath OS 2200

Note: *The Repository for ClearPath OS 2200 was formerly known as UREP, which remains the installation and support name.*

Internationalized Repository for ClearPath OS 2200

- Is 8-bit transparent.
- Can work with any coded character set (CCS) that is ASCII-like.
- Supports 8-bit data storage and retrieval directly; no special processing is necessary.

8.22. Service Library (SLIB)

Internationalized SLIB

The following SLIB services enable these I18N-related operations:

SSDFI and S$SREAD

Identify coded character sets (CCS) and read symbolic data format (SDF) file and element images in all CCSs.

S$SREAD

- Identifies CCSs and reads terminal images in all CCSs.
- Reads and processes SDF file images, SDF element images, and terminal images in ASCII-like CCSs the same way as ASCII images.

S$SWRITE

- Identifies CCSs and writes SDF file images, SDF element images, and terminal images in all CCSs.
- Processes and writes SDF file images, SDF element images, and terminal images in ASCII-like CCSs the same way as ASCII images.

SLIB Services and Their I18N Tasks

Symbolic Read Service (S$SREAD)

S$SREAD allows the callers for extended mode PLUS and for the C Compiler to read the following, one image at a time: SDF files, SDF elements, alternate symbiont files, and terminal images. The CCS of each element is returned to the calling program in the function packet.

- When TRANSLATE_MODE in the function packet is set to Translate, the calling program receives images in ASCII and ASCII-like CCSs.
- When TRANSLATE_MODE in the function packet is set to Do_Not_Translate, the calling program receives images in all CCSs.

Symbolic Write Service (S$SWRITE)

S$SREAD allows the callers for extended mode PLUS and the C Compiler to write the following, one image at a time: SDF files, SDF elements, alternate symbiont files, and terminal images. The calling program provides the CCS of each image in the function packet.

- When TRANSLATE_MODE in the function packet is set to Translate, the calling program writes images in ASCII and ASCII-like CCSs.
- When TRANSLATE_MODE in the function packet is set to Do_Not_Translate, the calling program writes images in all CCSs

8.23. Shared File System (SFS 2200)

Internationalized SFS 2200

- Supports 8-bit data storage and retrieval.
- Includes enhanced DSDF and MSAM file functions.

Support for 8-Bit Data Storage and Retrieval

SFS 2200 is 8-bit transparent and will work with any coded character set (CCS) that is ASCII-like. It supports 8-bit data storage and retrieval directly; no special processing is necessary.

Enhanced DSDF and MSAM File Functions

For applications developed using the COBOL Compiler, SFS 2200 allows you to

- Identify the coded character set (CCS) used to represent the data in an DSDF file.
- Specify the sorting sequence used for the keys in an MSAM file.

8.23.1. Identifying CCSs in DSDF Files

Background

For a new DSDF file, you can indicate the CCS identifier on the CODE TYPE clause of the OPEN FILE FOR OUTPUT statement.

Process

For an existing DSDF file, the COBOL Compiler can do one of the following:

- Store the value of the CCS identifier within the variable specified in the CODE TYPE clause of the file for which the OPEN INPUT statement is being performed, **or**
- Store the value of the variable specified in the CODE TYPE clause within the file header during execution of the OPEN OUTPUT statement.

Sample Code for DSDF Files

```
      *
      * Identify the CCS for data in a DSDF file.
      *
      * Declare the file.
      *
           SELECT fl ASSIGN TO A-SHARED-FILE
                     ORGANIZATION RELATIVE
                     CODE TYPE IS file-ccs.
      *
      * Declare the variable file-ccs.
      *
       01 file-ccs    PIC 99 BINARY.
                 .
                 .
                 .
      *
      * Get CCS when opening existing file.  It will be placed
      * in the CODE TYPE variable when the file is opened.
      *
           OPEN INPUT fl.
           IF file-ccs NE 35
              DISPLAY 'Code Type is not ISO 8859-1' UPON PRINTER
              STOP RUN.
      *
      * Establish CCS when opening new file for output.  The
      * CCS of the file will be set to the value of the variable
      * specified in the CODE TYPE clause of the file being opened.
      *
           MOVE 35 TO file-ccs.
           OPEN OUTPUT fl.
```

or

```
           CALL 'UCOB$CNM2CNB' USING isl-status, 'ISO8859-1',
              ccs-number-format, file-ccs.
           OPEN OUTPUT fl.
                 .
                 .
                 .
```

8.23.2. Specifying Sorting Sequence for Keys in MSAM Files

Background

For binary sorting order, SFS 2200 supports binary collation of 8-bit data directly. No special processing is necessary.

Procedure

For regional or national sorting order, follow these steps:

1. Create two key fields, one with the displayable text in it and one with an ordering key based on the displayable text.
2. Use the STRING_TRANSFORM service routine to create the ordering key.
3. Include the name of the locale on the OPEN FILE statement. The COBOL Compiler changes the locale name into its corresponding locale number, then writes the locale number into the file header record.

Sample Code for MSAM Files

```
*
* Identify the locale for data in an MSAM file.
*
* Declare the file.
*
     SELECT fl ASSIGN TO A-SHARED-FILE
               ORGANIZATION IS INDEXED
               LOCALE IS locale-name.
*
* Declare the variable locale-name.
*
 01 locale-name    PIC X(16).
              .
              .
              .
*
* Get locale when opening existing file.
*
     OPEN INPUT fl.
     IF locale-name NOT = 'en_US.ASCII'
        DISPLAY 'Locale is not en_US.ASCII' UPON PRINTER
        STOP RUN.
*
* Establish locale when opening new file for output.
*
     MOVE "en_US.ASCII" TO locale-name.
     OPEN OUTPUT fl.
             .
             .
             .
```

8.24. Sort/Merge Utility

Internationalized Sort/Merge Utility

- Calls I18NLIB service routines internally.
- Uses sort or merge operations in extended mode environment.
- Contains enhanced processor and subroutines to order data by locale.

Internal Calls to I18NLIB Service Routines

The I18NLIB service routines cannot be called directly from the Sort/Merge Utility. Settings for the I18N parameters allow the SORT processor or subroutines to call them internally.

I18N Sort/Merge Operations

An I18N sort or merge operation is one that contains an international key. This operation is

- Restricted to sorting in an extended mode environment.
- Performed by using the C Compiler, the COBOL Compiler, a basic mode MASM program, or an extended mode PLUS program.
- Performed by using the SORT processor.

For the Sort/Merge Utility to execute properly in the I18N environment, the scratch file sizes for extended mode disk sorts must be increased to larger than 36 tracks.

Enhanced Processor and Subroutines

The Sort/Merge Utility provides interfaces for the C Compiler and for extended mode PLUS. These interfaces are described in detail in the *Sort/Merge Programming Guide*.

The SORT processor and subroutines have been enhanced to order the data according to the locale specified or the locale in force at the time of entering the sort subroutines. For more details about locale name and values, go to Related Topics.

8.24.1. Configuration Parameters

CCSLOCNUM Configuration Parameter

This parameter lets you control the setting of the locale number and code-type numbers for files created by the SORT processor.

Constraints

The effect of this parameter can be overridden by the LOCALE and CODE-TYPE processor parameters.

Values

The CCSLOCNUM settings are PREI18N, DEFAULT, or INPUTDATA. The default setting is DEFAULT.

If the setting is	The SORT processor
PREI18N	Writes the same value in the code-type field or locale field that was written in Sort/Merge Utility levels before level 19R1.
DEFAULT	Uses DEFLOCALE configuration parameter to determine what should be written to the file.
INPUTDATA	Sets the code-type and locale fields based on the code type and/or locale of the input files.

CCSLOCNUMCHK Configuration Parameter

This parameter determines if the SORT processor should perform code-type and locale conflict checks.

Values

The CCSLOCNUMCHK settings are ON and OFF. The default value is ON.

If the setting is	The SORT processor
ON	Performs checks to insure that there are no conflicts between the code type and the locale settings that are being used.
OFF	Does not perform these conflict checks.

DEFLOCALE Configuration Parameter

This parameter defines the default locale name to be used for I18N sort or merge operations.

Constraints

The DEFLOCALE configuration parameter can be overridden by the LOCALE processor parameter.

Values

Locale names are case-sensitive. The locale name must contain from 1 to 16 characters, which may be ASCII characters in the range ! through ~ (041 through 0176). The "(042), ‘ (047) and / (057) characters cannot be used.

The predefined values are '_QUERY' (for the current locale) and '_DEFAULT' (for the system default locale). The default locale name for DEFLOCALE is _QUERY.

RBSIZE Configuration Parameter

This internal parameter defines the record bank size for an I18N sort or merge operation.

Constraints

In an I18N sort or merge operation, the SORT subroutines cannot translate the key in place within the record because the transformation may exceed the original key size. To accommodate this change, the SORT subroutines create a new bank, which is used to hold the original user records.

Values

The RBSIZE value must be within the range of 4194304 through 16777216. The default setting is 4194304.

8.24.2. Processor Parameters

8.24.2.1. DATA Processor Parameter

This parameter determines the character type of the data processed by the SORT processor. Enhancements to this parameter include two new specifications: DATA=6BIT and DATA=9BIT.

- DATA=6BIT is identical in meaning and functionality to DATA=FIELDATA.
- DATA=9BIT indicates that the character data contains 9-bit bytes, but this does not necessarily mean that the character data is ASCII.

Constraints

- The SORT processor does not convert this data.
- The DATA parameter is used to determine the number of characters per word only.

Format

```
DATA=character-type
```

Values

The valid field specifications for *character-type* are ASCII, 9BIT, and FIELDATA or 6BIT.

ASCII

Specifies that all character data is ASCII. This is the default setting if you do not specify the C or M option on the SORT processor call.

9BIT

Specifies that all character data is 9-bit character data. It does not mean that that character data is ASCII.

FIELDATA or 6BIT

Specifies that all character data is Fieldata. This is the default setting if you do specify the C or M option on the SORT processor call.

8.24.2.2. KEY Processor Parameter

This parameter tells the SORT processor that a certain part of each record is to be treated as a key field.

Format

KEY=*sc1*,*nc1*,*type1*[(*subtype1*)],*ad1*[,*fldn1*]:...:*sc64*,*nc64*,*type64*[(*subtype64*)],*ad64*[,*fldn64*]

Values

sc1, ..., sc64

are integers indicating the starting character of the key field. The first character position of a record is 1. The maximum allowable value for the start character is 16384 for 9-bit characters and 24576 for 6-bit characters.

nc1, ..., nc64

are integers indicating the length, in characters, of the key field. The SORT processor determines whether the characters are 9-bit or 6-bit by using the DATA parameter or the A, C, or M SORT processor call options.

type1, ..., type64

specifies the key type. Use the I key type to specify that an international sort or merge should be performed. This key contains 9-bit data that will be ordered according to the rules of the locale currently in force. See 8.24.2.4 for the complete list of key types.

subtype1, ..., subtype64

is an optional field specifying additional information about the key type. This field specification, valid only when the key type field is I, indicates if trailing spaces should be ignored or included when ordering the I key. Possible values are IGNORETSP to ignore trailing spaces or INCLUDETSP to include trailing spaces (the default).

ad1, ..., ad64

specifies the sort or merge sequence. The valid ad field specifications are A (ascending sequence) or D (descending sequence).

fldn1, ..., fldn64

are integers specifying the significance of the key fields from 1 to 64 (major to minor). If you use this field on one key field definition, you must use it on all key field definitions. If you do not specify fldn fields, the first key field defined is considered the major key and any subsequent key fields defined are minor keys.

8.24.2.3. KEYW Processor Parameter

This parameter tells the SORT processor that a certain part of each record is to be treated as a key field.

Format

KEYW=*sw1*,*sb1*,*nb1*,type1[(*subtype1*)],*ad1*[,*fldn1*]:...:*sw64*,*sb64*,*nb64*,*type64*[*subtype64*)],*ad64*[,*fldn64*]

Values

sw1, ..., sw64

are integers indicating the starting word number of the key field. The first word of the record is 1. The maximum allowable value for the start word is 4096.

sb1, ..., sb64

are integers indicating the starting bit number within the start word of the key field. The leftmost or most significant bit number is 35, and the least significant bit number is zero (0).

nb1, ..., nb64

is an integer that specifies the number of bits in the key field.

type1, ..., type64

specifies the link to key type definitions. Use the I key type to specify an international sort or merge operation. See 8.24.2.4 for the complete list of key types.

subtype1, ..., subtype64

is an optional field specifying additional information about the key type. This field specification, valid only when the key type field is I, indicates if trailing spaces should be ignored or included when ordering the I key. Possible values are IGNORETSP to ignore trailing spaces or INCLUDETSP to include trailing spaces (the default).

ad1, ..., ad64

specifies the sort or merge sequence. The valid ad field specifications are A for ascending sequence or D for descending sequence.

fldn1, ..., fldn64

are integers specifying the significance of the key fields from 1 to 64 (major to minor). If you use this field on one key field definition, you must use it on all key field definitions. If you do not specify fldn fields, the first key field defined is considered the major key and any subsequent key fields defined are minor keys.

8.24.2.4. KEY/KEYW Key Field Types

Key Type	Definition
A	Fieldata alphanumeric
B	Signed binary
D	Fieldata sign-leading separate decimal
DATE	ASCII alphanumeric containing a date when the first 18 bits represents a year
G	Fieldata sign-trailing separate decimal
I	Contains 9-bit data ordered according to the rules of the locale; indicates that an I18N sort or merge operation should be performed.
J	Signed packed decimal
L	Fieldata sign-leading overpunched decimal
M	IBM fixed-point binary
P	Fieldata sign-trailing overpunched decimal
Q	ASCII sign-trailing overpunched decimal
R	ASCII sign-leading separate decimal
S	ASCII alphanumeric
T	ASCII sign-leading separate decimal
U	Unsigned binary
V	ASCII sign-leading overpunched decimal

8.24.2.5. LOCALE Processor Parameter

This parameter tells the SORT processor which locale to use for I18N sorting or merging. Only one LOCALE parameter can be specified.

In the Unisys implementation of I18N, each locale name has a corresponding locale number. This number is written in the header block of an MSAM file.

Format

```
LOCALE='locale-name'
or
LOCALE="locale-name"
or
LOCALE=locale-name
```

Values

This *locale-name* value is the name of the locale. It may be delimited by either single or double quotation marks. Whichever mark is used to begin the name string must also be used to end it.

- The locale name consists of 1 to 16 characters from the ISO 8859.1 code set and is case-sensitive. No verification is made to determine whether this is the code set being used to enter the parameter.
- The locale name can be either '_QUERY' (for the current locale) or '_DEFAULT' (for the system default locale). Both options are case sensitive.

8.24.2.6. CODE-TYPE Processor Parameter

This parameter tells the SORT processor what character set to write in the header of an SDF, DSDF, or mass storage alternate symbiont file created by the Sort/Merge Utility.

This parameter is local to the FILEOUT parameter and has meaning for all operations (COPY, MERGE, SORT, TAG, RETRIEVE, and TAG/RETRIEVE).

Format

```
CODE-TYPE=code-type
```

Value

This is an integer value, which must be 0 through 077. If the first number is 0, the SORT processor assumes it is an octal number. If the first number is not 0, a decimal number is assumed.

8.24.3. Subroutine Parameters (MASM Programs Only)

8.24.3.1. KEY Subroutine Parameter

This parameter specifies a key field to be used for an I18N sort or merge operation.

Format

```
'KEY', char-pos, number-of-char, 'type[(subtype)]',[, ['order']
[,field-number]]
```

Values

char-pos

is an integer specifying the starting character of the key stated in either 6-bit or 9-bit characters. The first character position of a record is 1. For key types A, B, D, G, L, M, P, or U; the field is 6-bit characters, and for key types DATE, I, J, Q, R, S, T, or V, the field is 9-bit characters. See 8.24.2.4 for the complete list of key types.

number-of-char

specifies the length of the key in either 6-bit or 9-bit characters, depending on the type of key.

type

specifies the key field type. Use the I key type to specify an I18N sort or merge operation. See 8.24.2.4 for the complete list of key types.

subtype1, ..., subtype64

is an optional field specifying additional information about the key type. This field specification, valid only when the key type field is I, indicates if trailing spaces should be ignored or included when ordering the I key. Possible values are IGNORETSP to ignore trailing spaces or INCLUDETSP to include trailing spaces (the default).

order

is an optional field. If you specify order, you also need to specify either A for an ascending sort/merge (the default) or D for a descending sort/merge.

field-number

is an integer between 1 and 64 that indicates the significance of the key field. The most significant (major) key is number 1, and the least significant (minor) is number 64.

- If you specify field-number for any key field, you must specify it for all key fields.
- If you do not specify field-number, the first key field defined is the major key and any subsequent key fields defined are the minor keys.

8.24.3.2. KEYW Subroutine Parameter

This parameter specifies a key field for an I18N sort or merge operation.

Format

```
'KEYW',word-pos,bit-pos,number-of-
bits,'type[(subtype)]'[,['order'],[,field-number]]
```

Values

word-pos

must be the starting word number of the key field. The first word within a record is number 1.

bit-pos

indicates the most significant bit of the key field within the first word of the key. The most significant bit position is number 35, and the least significant is number 0.

number-of-bits

specifies how many bits are in the key field.

type

specifies the key field type. Use the I key type to specify an I18N sort or merge operation. See 8.24.2.4 for the complete list of key types.

subtype1, ..., subtype64

is an optional field specifying additional information about the key type. This field specification, valid only when the key type field is I, indicates if trailing spaces should be ignored or included when ordering the I key. Possible values are IGNORETSP to ignore trailing spaces or INCLUDETSP to include trailing spaces (the default).

order

is an optional field. If you specify order, you also need to specify either A for an ascending sort/merge (the default) or D for a descending sort/merge.

field-number

is an integer between 1 and 64 that indicates the significance of the key field, with 1 being the major key field and 64 being the minor field.

- If you specify field-number for any key field, you must specify it for all key fields.
- If you do not specify field-number, the first key field defined is the major key and any subsequent key fields defined are the minor keys.

8.24.3.3. LOCALE Subroutine Parameter

This parameter specifies the desired locale for a sort or merge operation. Only one LOCALE parameter can be specified.

If an I18N sort or merge operation is requested (presence of an I key) and a LOCALE parameter is not specified, the locale currently in force will be used.

Format

```
LOCALE,'locale-name'
```

Value

The value of *locale-name* is a string that indicates the desired locale for the sort or merge operation. A locale name is from 1 to 16 characters long and is case-sensitive.

8.24.4. Sample Runstream

The following runstream performs an internationalized sorting operation by ordering the data contained in the input file DATA1 in ascending order, according to the collation rules specified in the de_DE.8859-1 locale (specifies German language as used in Germany).

Notice the use of the CODE-TYPE and LOCALE parameters and the specification of the key type in the KEY parameter.

```
@SORT,E
FILEIN=DATA1 MODE=SDF
FILEOUT=OUT MODE=SDF CODE-TYPE=043
RSZ=80,CHARACTERS
KEY=1,5,I,A
```

```
CORE=500000
LOCALE='de_DE.8859-1'
@EOF
```

8.24.5. Sample Code

This code example in basic mode MASM shows how an internationalized sorting operation can be performed.

```
          axr$
          $include  'rproc$/'         .
pf        $form     12,6,18           .
$(1),start                            .
          r$opncb,dbanksize,dbank params . initialize sort subroutines
release                               .
          r$ead     (sort,buf)        .
          r$relcb,dbank  (rsz,buf)    . common bank linkage
          j         release           .
sort                                  .
          p$rint    (pf 1,msg1l,msg1) .
          r$sortcb,dbank              . common bank linkage
retrieve                              .
          r$retcb,dbank eof           . common bank linkage
          lxi,u     a0,0              . normal return 1,x11
          aa        a0,((01*/24)++(rsz*/18)) .
          er        print$            .
          j         retrieve          .
eof                                   .
          p$rint    (pf 1,msg2l,msg2) .
          er        exit$             .
$(2)                                  .
rsz       $equ      13                .

.
. Notice the use of the I key type in the KEY R$FILE parameter
. and the LOCALE R$FILE parameter in the following parameter
. table.
.

params    r$file                      ;
          'rsza',rsz*4                ;
          'key',1,20,'i'              ;
          'core',500000               ;
          'locale','POSIX'            ;
          'files','xc','xd','xe'      ;
          'fpoc',release              ;
          'lpoc',retrieve
dbank     $res      02200             .
dbanksize $equ      $-dbank           .
buf       $res      rsz               .
msg1      'all records have been released to sort.' .
```

```
msg1l     $equ      $-msg1            .
msg2      'SORT end of file exit taken.' .
msg2l     $equ      $-msg2            .
          $end      start             .
```

8.24.6. Sorting and Collating within I18N Applications

Data can be sorted and collated from a number of different places within an I18N application. You can use the following service routines, compiler products, and database products to perform these operations. You can also use the I18N version of the Sort/Merge utility. More about the Sort/Merge Utility (SORT)

Service Routines

STRING_COLLATE compares two character strings, based on the rules in the locale definition file. This service routine can be called directly from the application.

STRING_TRANSFORM changes a character string to an ordering key, which is then used in a binary comparison. This service routine can be called directly from the application.

Compiler Products

COBOL Compiler

USAGE IS DISP-I18N clause is used to call the STRING_COLLATE service routine for comparing two data items. The SORT statement for the COBOLCompiler supports sorting of data items defined by this clause. Sorting is based on the locale identified by the LC_COLLATE category, which is established by the SETLOCALE service routine.

USAGE IS BYTE-I18N clause identifies a data item to hold the ordering key returned by the STRING_TRANSFORM service routine. The COBOL Compiler use this ordering key to perform a binary comparison.

IF statement is used to compare character strings.

C Compiler

strcoll() function is used for locale-specific sorting.

IF statement is used to perform a binary comparison of character strings, as required by the C standard.

FORTRAN Compiler

FORTRAN application must make an interlanguage call to the COBOL Compiler STRING_COLLATE service routine to compare two character strings.

FORTRAN compiler always generates a binary comparison for these character strings.

Database Products

Network Database Server

Locale-specific ordering is performed on sorted chain sets and index sequential record placement.

LOCALE clause on SCHEMA name statement identifies the locale to use for ordering the keys.

Relational Database Server

COLLATE clause on a column definition indicates the locale to be used to order data in the column.

Locale-specific ordering of column values is performed within a table.

PCIOS and SFS 2200

- SSDF or DSDF files

  The coded character set (CCS) for the file images is identified in the file header by the CODE TYPE clause on the OPEN FILE statement for the COBOL Compiler. The application must use the STRING_COLLATE service routine to perform locale-specific comparisons of data from the file.

- MSAM files

  The locale used to create ordering keys is identified by the LOCALE clause on the OPEN FILE statement for the COBOL Compiler. The application must use the STRING_TRANSFORM service routine to create the ordering keys and write them to the file.

8.24.7. Error Messages

SORT Processor Error Messages

<56> DATA TYPE INVALID FOR THE OUTPUT FILE MODE

When the mode of the output file is	The data type associated with this operation must be
ANSI	ASCII or 9 bit
CFH	Fieldata or 6 bit

<76> KEY FIELD TYPE IS INCONSISTENT WITH THE DATA TYPE

One of the following invalid key types was used:

- An ASCII or 9-bit key type was used with DATA=FIELDATA or DATA=6BIT,
- A Fieldata or 6-bit key type was used with DATA=ASCII or DATA=9BIT.

<82> PARAMETER NOT ALLOWED WITH FILEOUT

The specified parameter is not supported by FILEOUT. Valid parameters are: BLOCK, BMBLOCK, CODE-TYPE, EQUIP, FILEID, LABEL, MODE, PREP, REELS, REVMSAMBLK, and SPLIT.

<131> AN I KEY MUST NOT BE SPECIFIED FOR THIS OPERATION

An I key type is meaningful only for a sort or merge operation--not for a TAG, TAG/RETRIEVE, or RETRIEVE operation.

<132> WARNING - A CODE-TYPE DOES NOT EXIST FOR THIS MODE

The CODE-TYPE parameter should only be specified for files with a mode of CARDS, PRINT, SDF, or DSDF. These are the only file modes for which a code type has meaning.

<133> WARNING - CODE-TYPE EXCEEDS THE MAXIMUM ALLOWED VALUE

The CODE-TYPE value exceeded the maximum allowed value of 63 (octal 077). A code type 01 will be assumed.

<134> CODE-TYPE SPECIFIED IS A UNISYS RESERVED CHARACTER SET VALUE

A Unisys reserved character set value was specified in the CODE-TYPE parameter. You need to update this parameter and rerun the program.

<135> DATA PARAMETER CONFLICTS WITH OTHER PARAMETERS OR OPTIONS

The value specified on the DATA parameter conflicts with another parameter or with a processor call option.

If you specify this DATA parameter	Conflicts can occur if you include a (an)
DATA=FIELDATA or DATA=6BIT	I key
	CODE-TYPE parameter with a value not equal to zero
	LOCALE parameter
DATA=ASCII or DATA=9BIT	CODE-TYPE parameter with a value equal to zero
	Processor call option C or M

<136> WARNING - THE LENGTH OF THE LOCALE NAME IS INVALID

The name specified on the LOCALE parameter contains an incorrect number of characters.

- The locale name must contain at least 1 character but not more than 16 characters.
- If the locale name contains more than the maximum number allowed, it is truncated to the maximum number (16).

<137> I KEY NOT ALLOWED WITH THIS MODE

An I key can only be specified when the mode of the output file is CARDS, PRINT, SDF, DSDF, MSAM, or ANSI.

<140> THE CODE-TYPE CONFLICTS WITH THE INPUT FILE (S) CODE TYPE

A conflict exists between the code type specified on the FILEOUT parameter and the code type of the input files. The code type of the input file must either be 01 or match the code type specified on the FILEOUT parameter. The error could be caused by the setting of the DEFCTYPE configuration parameter.

<141> I KEY TYPE CANNOT BE USED WITH A BASIC MODE SORT

An I key type can be used only with an extended mode sort.

<142> WARNING - CODE-TYPE CONFLICTS WITH THE CODE TYPE OF LOCALE

The value specified in the CODE-TYPE parameter does not match the code type associated with the current locale setting. Therefore:

- The value specified in the CODE-TYPE parameter is ignored.
- Any CODE-TYPE number written to the output file will be the code type associated with the locale in force for this operation.

<143> LOCALE CONFLICT WITH MSAM FILES

There is a conflict between the locales for the MSAM input and output files. This error could be caused by

- The existence of multiple MSAM input files that do not have the same respective locales.
- The setting of the DEFLOCALE configuration parameter.

Setting the CCSLOCNUMCHK configuration parameter to OFF will prevent this error.

<144> CODE-TYPE OF AN INPUT FILE AND MSAM OUTPUT FILE CONFLICT

There is a conflict between the code type associated with the locale for the MSAM output file and the code type of a non-MSAM input file. This error could be caused by the setting of the DEFLOCALE configuration parameter.

Setting the CCSLOCNUMCHK configuration parameter to OFF will prevent this error.

<145> ERROR RETURNED BY I18NLIB SERVICE ROUTINE

An I18NLIB service routine encountered an error that the SORT processor did not expect. This could be an internal SORT error. Ask your system administrator to submit a UCF.

<146> LOCALE NOT FOUND

The specified locale could not be found on the system. Locale names are case-sensitive. You need to verify the following:

- Is the specified locale name correct?
- Is the specified locale installed on your system?

SORT Subroutines Error Messages

Table 8–9. SORT Subroutines Error Messages

Extended Mode Linkage	Basic Mode Linkage	Description
44592	<WO>	ERROR RETURNED BY SERVICE ROUTINE - An I18NLIB service routine encountered an error that the SORT processor did not expect. This could be an internal SORT error. Ask your system administrator to submit a UCF.
44593	<W1>	LOCALE NOT FOUND - The locale specified for the sort or merge operation could not be found on the system. You need to verify the following: • Is the specified locale name correct? • Is the specified locale installed on your system?
44594	<W2>	I KEY NOT ALLOWED WITH DATA REDUCTION OR USER COMPARE ROUTINE - An I key type is not allowed when a data-reduction own code (DROC) or a user compare (COMP) routine is specified.
44595	<W3>	I KEY TYPE CANNOT BE USED WITH BASIC MODE SORT - An I key type can only be used with an extended mode sort.
44596	<W4>	INSUFFICIENT SCRATCH FILES FOR I18N SORT - The scratch files are not large enough for this I18N sort operation. The scratch file size should be larger than 36 tracks.
44597	<W5>	INSUFFICIENT RECORD BANK FOR CORE ONLY I18N SORT - The record bank is not large enough for this I18N core-only sort. You need to do one of the following: • Increase the size of the RBSIZE configuration parameter. • Change the operation to an extended mode disk sort.

8.25. Symbolic Stream Generator (SSG)

Internationalized SSG

SSG processes images of all coded character set identifiers (CCS ID).

- All nonzero image types are assumed to be ASCII.
- All zero type images are assumed to be Fieldata.

8.26. System Service Routine Library (SYSLIB)

8.26.1. Internationalized SYSLIB

This SYSLIB definition	Enables this I18N-related operation
SYS$DEF	Identifies which coded character sets (CCS) are ASCII-like.
SDFI SAR$ READ SIR$	Identify CCSs and read system data format (SDF) file and element images in all CCSs.
SAR$ READ SIR$	Identify CCSs and read runstream (terminal) images in all CCSs. Read and process SDF file images, SDF element images, and runstream images in all ASCII-like CCSs the same way as ASCII.
SDFO SAR$ WRITE SIR$ SOR	Identify CCSs and write SDF file and element images in all CCSs.
SAR$ WRITE SIR$	Identify CCSs and write runstream images in all CCSs. Write and process runstream images in all ASCII-like CCSs the same way as ASCII images.
SAR$ WRITE SIR$ SOR	Write and process SDF file images and SDF element images in all ASCII-like CCSs the same way as ASCII images.
SAR$ COM	Outputs console messages in all ASCII-like CCSs the same way as ASCII messages.

8.26.2. SYSLIB Definitions and Routines and Their I18N Tasks

Select SYS$DEF or a SYSLIB routine to review the specific I18N tasks it performs.

System Definitions (SYS$DEF)

ASCLIKE$ is a 72-bit ASCII-like mask that is available to Basic Mode MASM callers.

- If CCS*n* is defined as ASCII-like, bit 71-*n* is set to 1.
- If CCS*n* is not defined as ASCII-like, bit 71-*n* is set to zero (0).

System Data Format Input Routine (SDFI)

SSDFI allows the callers for extended mode PLUS as well as the C, COBOL Compiler to read SDF files and elements one image at a time. The CCS of the images are identified in certain types of SDF control records and data records. If these records are present, this data is returned to the calling program.

See "List of Control Record Types" and "List of SDF File Types" in Section 2.

System Data Format Output Routine (SDFO)

SDFO allows Basic Mode MASM callers to write SDF files and elements one image at a time. The CCS of the images are identified in two types of SDF control records. The calling program formats these records in word 10 of the SDFO packet.

Symbolic Access Routine Read (SAR$ READ)

SAR$ READ allows Basic Mode MASM and PLUS callers to read the following, one image at a time: SDF files, SDF elements, alternate symbiont files, and terminal images. The CCS of each image is returned to the calling program in the function packet.

- When UNTRANSLATE_MODE in the function packet is set to OFF, images in ASCII and ASCII-like CCSs are returned to the calling program.
- When UNTRANSLATE_MODE in the function packet is set to ON, images in all CCSs are returned to the calling program.

Symbolic Access Routine Write (SAR$ WRITE)

SAR$ WRITE allows Basic Mode MASM and PLUS callers to write the following, one image at a time: SDF files, SDF elements, alternate symbiont files, and terminal images. The calling program provides the CCS of each image in the function packet.

- When UNTRANSLATE_MODE in the function packet is set to OFF, the calling program can write images in ASCII and ASCII-like CCSs.
- When UNTRANSLATE_MODE in the function packet is set to ON, the calling program can write images in all CCSs.

Symbolic Access Routine Com (SAR$ COM)

SAR$ COM allows Basic Mode MASM and PLUS to output operator console messages. The calling programs can output images in ASCII or ASCII-like CCSs, but all are treated as if they are ASCII images.

Symbolic Input/Output Routine (SIR$)

SIR$ is available to Basic Mode MASM callers. The caller must be a processor that has been initialized previously with a call to one of the SYSLIB preprocessor routines (PREPRO, PREPRM, or PREPF$).

SIR$ Processing

In the following process, source input and source output refer to an SDF file or element.

1. SIR$ can provide the required output either by reading the source input or by merging it with runstream (terminal) correction images.
2. It can then write updated source output (OPTIONAL).

Reading Merged Input and Returning Images

SIR$ allows merged input to be read one image at a time. SIR$ can be initialized to operate in these ways:

- The CCS of each image is returned.
- Only images in Fieldata, ASCII, and ASCII-like CCSs are returned.
- Images in all CCSs are returned.

Determining CCS of Written Source Output Images

The CCS of written source output images is based on

- The CCS of the source input image.
- The option characters that are included when the processor is called. These characters are included as

```
@MASM,SVPQ TPF$.TST
```

Symbolic Output Routine (SOR)

SOR is available to Basic Mode MASM callers. The caller must be a processor that has been initialized previously with a call to one of the SYSLIB preprocessor routines (PREPRO, PREPRM, or PREPF$).

Calling SOR

SOR is usually called by processors that produce symbolic element output, rather than omnibus, relocatable, absolute, or object module element output.

Specifying CCSs

SOR allows the calling program to specify the CCS of each image to be written.

Writing Images

The calling program can write images in all CCSs.

Section 9
Installing and Configuring I18NLIB Features and I18N Products

9.1. Currently Available Locales

The following locale names (column 1) and corresponding numeric identifiers (column 4) are used by the locale-name-and-number service routines.

Note: *This list includes only those locales that are supported by the I18NLIB service routines; it does not include all locales that have been developed by the International Standards Organization (ISO).*

Locale Name	Language/Where Used	Associated Charmap	Locale Number
C	C standard	ISO8859-1	1
da_DK.8859-1	Danish/Denmark	ISO8859-1	2
de_AT.8859-1	German/Austria	ISO8859-1	3
de_CH.8859-1	German/Switzerland	ISO8859-1	4
de_DE.8859-1	German/Germany	ISO8859-1	5
de_LU.8859-1	German/Luxembourg	ISO8859-1	6
el_GR.8859-7	Greek/Greece	ISO8859-7	7
en_CA.8859-1	English/Canada	ISO8859-1	8
en_DK.8859-1	English/Denmark	ISO8859-1	9
en_GB.8859-1	English/Great Britain	ISO8859-1	10
en_IE.8859-1	English/Ireland	ISO8859-1	11
en_US.8859-1	English/USA	ISO8859-1	42
es_ES.8859-1	Spanish/Spain	ISO8859-1	13
euro.8859-1	Europe/ISO 8859-1 suite	ISO8859-1	43
fr_BE.8859-1	French/Belgium	ISO8859-1	17
fr_CA.8859-1	French/Canada	ISO8859-1	18
fr_CH.8859-1	French/Switzerland	ISO8859-1	19
fr_FR.8859-1	French/France	ISO8859-1	20

Locale Name	Language/Where Used	Associated Charmap	Locale Number
fr_LU.8859-1	French/Luxembourg	ISO8859-1	21
it_IT.8859-1	Italian/Italy	ISO8859-1	25
nl_BE.8859-1	Dutch/Belgium	ISO8859-1	31
nl_NL.8859-1	Dutch/Netherlands	ISO8859-1	32
no_NO.8859-1	Norwegian/Norway	ISO8859-1	33
pl_PL.8859-2	Polish/Poland	ISO8859-2	34
POSIX	POSIX standard	ISO8859-1	35
pt_PT.8859-1	Portuguese/Portugal	ISO8859-1	36

9.2. Updating and Installing the I18NLIB Run-Time Tables

The I18NLIB run-time tables contain processed locale definition files, FORMMAPs, and transliteration sources (CCS-TO-CCS elements) that are used by the I18NLIB service routines. These tables normally do not require updating because they are released containing all supported locales, FORMMAPs, and CCS-TO-CCSs. However, sites may want to

- Modify a transliteration source element to include private use (user-defined) characters
- Remove unused locales, FORMMAPs, or CCS-TO-CCS conversions

Note: *Removing any items must be done with care because they might be needed internally by products that use I18NLIB.*

Update Guidelines

The default location for sources used to create run-time tables is in SYSLIB*I18NLIB-LANG. Any local updates to any element must be placed in this file, or another file must be created that contains all the sources in the SYSLIB*I18NLIB-LANG file, plus the updated source elements.

Note: *The use of another file is recommended so that updates are easier for moving up levels of I18NLIB.*

The system administrator usually configures or updates run-time tables because additional privileges may be required to do so and because all I18NLIB users on a system will be affected by the updates. If run-time tables are reconfigured or updated with site-specific information, the same steps must be performed after each installation of I18NLIB, preferably immediately after the installation is completed.

Note: *Please be aware of any potential updates that Unisys may make when moving up levels of I18NLIB.*

Restriction

Certain locales and CCS conversions cannot be removed because they are required by default or by other selected items.

Procedure for Updating and Installing Run-Time Tables

To update and install the I18NLIB run-time tables, you need to first generate the updated run-time tables, then check the breakpoint print file for errors, and finally, install the updated run-time tables.

1. Create a separate file (such as SITE*I18NLIB-LANG) to contain the updates to the run-time tables.
   - Since the SYSLIB*I18NLIB-LANG file is updated during each installation of I18NLIB, creating this separate file will prevent your local updates from being overwritten..
   - Also note that when Unisys releases new levels of the I18NLIB software, there may be modifications to the source code that you will need to transfer to your local file.
2. Execute the RTTB processor by entering @SYSLIB*I18NLIB.RTTB.

   The RTTB processor produces a series of screens requesting LOCALEs, FORMMAPs, and CCS2CCSs to be generated.

3. When each query screen appears, select one of the following options:

ALL	For all items on the screen (see Note)
<number>	For one or more specific items (a list of selected items by number)
<Xmit>	For none

Note: *The released version has all of the tables generated ('ALL' response to each query screen).*

4. When these screens are completed, you are queried for the following additional information (recommended responses in bold type):
   - Name of file that contains the language definitions: SITE*I18NLIB-LANG
   - Application where Repository information will be placed
   - LOCDEF options for controlling listings: N or S
   - If you want debug mode enabled: N (unless indicated otherwise by Unisys)

     If you want to see the actual set of queries from the RTTB processor, see below.

     Once you have entered all the required information, the RTTB processor starts a batch run that generates the new run-time tables. Once this run is completed, proceed to step 5.

5. Look at the breakpoint print file (RTTB$BRKPT) from the batch run to verify that no errors have been encountered.

   This file can be quite large, as many warning messages are generated, but there should be NO errors generated.

   - If local modifications were made to any source, examine the file carefully in these locations to make sure no undesirable warning messages have been generated.
   - If there are no errors, the common bank absolute for the updated run-time tables is located in the element RTTB$FILE.RUNTIMETBL.

6. Save the active run-time tables for recovery, install the updated table, and activate the new run-time table.

   - If the common bank for the run-time tables was generated successfully, you can update the active run-time tables on your system. To perform this step, you must have write access to the file SYSLIB*I18NLIB-RTTA as well as common-bank reload privileges.
   - If I18NLIB is being used by any program, you should not install updated run-time tables.
   - Use the following commands to complete step 6:

```
@CAT,P   SITE*I18NLIB-RTTA.,F///999.
@ASG,A   SITE*I18NLIB-RTTA
@COPY SYS$LIB$*I18NLIB-RTTA.,SITE*I18NLIB-RTTA.
@COPY,SROA    RTTB$FILE.,SYSLIB$*I18NLIB-RTTA.
@PACK    SYS$LIB$*I18NLIB-RTTA.
@SYS$LIB$*SOLAR.RELOAD    0405234
@FREE    SITE*I18NLIB-RTTA.
```

   The updated run-time tables will now be installed and will be used on subsequent calls to the I18NLIB service routines.

7. Perform a LIBSAVE of the new library/system environment.

   This step is needed in case a LIBLOAD is required later to restore the new updated run-time tables.

See Also: Translating Private Use Characters from One CCS to Another in *Introduction to Coded Character Sets (CCS)* and *Installing I18N Products That Use ELMS* in *Extended Language Message Systems (ELMS).*

Queries from the RTTB Processor

The following is the set of queries asked by the RTTB processor after the queries for locales, FORMMAPS, and CCS2CCS sources. User input appears in **bold** text.

```
>Enter the name of the file (Q*FILE) containing language definitions.
><XMIT> to use the defined default file (SYS$LIB$*I18NLIB-LANG).
>File name? :>SITE*I18NLIB-LANG

>If you would like LOCDEF to put Repository information into a specific
>application, enter that name or simply <XMIT> to use the system-defined
```

```
>default.
>Application name? :>

>Enter the LOCDEF options you want for controlling listings:
> N – No listings (this is the default).
> S – Display source listings.
> T – Display the generated table.
> L – Same as the "S" and "T" options together.
>Enter LOCDEF options (<N>) :>

>Would you like to enable debug mode (<N> |Y? :>

>The runstream in RTTB$GEN.ADD will be @STARTed with a default run-id.
>Verify the results when complete in print file RTTB$BRKPT.
>
>END RTTB  hhmm:ss
```

9.3. Virtual Machine for the Java Platform on ClearPath OS 2200: I18N Requirements

Unisys OS 2200 Series systems are already able to handle 6-, 8-, and 9-bit character representations, but with the release of the Virtual Machine for the Java™ Platform on ClearPath OS 2200, 16-bit capabilities are included as well. This 16-bit coded character set (CCS) is known as Unicode or the UCS-2 CCS.

Requirements

This product requires an I18NLIB service routine to obtain the character size in bits for any supported character code. The NAME_AND_NUMBER packet interface routine provides this capability and returns the appropriate information in a user-defined packet area.

Appendix A
I18NLIB Error Statuses and Messages

This appendix lists the error code, error severity, error message, and a description of the errors detected and returned by I18NLIB for all language bindings other than C. The C error status is returned in the *errno* global variable and these statuses are defined in Section 3.

The error structure returned by I18NLIB is an ELMS error message number format. The error structure format consists of the error severity, the ELMS component ID (for I18NLIB the component id is ISL (023154), and the error number.

I18NLIB error statuses have three (3) severity levels:

0 Success — The service routine was successful.

3 Warning — The service routine completed but detected a condition that required an assumption to be made or an alternate result returned.

7 Failure — The service routine detected an error situation which prevents the successful completion of the function. The resultant values are either undefined or a partial result is returned along with an indication of the location of the error.

Table A–1 lists the errors in numerical order along with the severity of the error, the defined message text, and an explanation of the error and a potential resolution.

Table A–1. List of Error Statuses

Message Text	Error Number	Severity Code	Description of Error Status
OK	0	0	The operation was successful.
INVALID PACKET LENGTH	1	7	The basic mode MASM packet length specified was invalid.
INVALID PACKET VERSION	2	7	The version specified in the basic mode MASM packet was invalid.
BUFFER TOO SHORT	3	7	The service routine tried to write beyond the end of the buffer provided by the caller.
FROM CCS NOT DEFINED	4	7	No transliteration table is available for the CCS to be transliterated.

Table A–1. List of Error Statuses

Message Text	Error Number	Severity Code	Description of Error Status
TO CCS NOT DEFINED	5	7	No transliteration table is available for the resulting CCS.
ENCODING NOT SUPPORTED	7	7	The specified encoding is not supported.
ENCODING OF TO BUFFER FAILED	8	7	The encoding specified for to-buffer could not be successfully accomplished.
BAD CHAR	13	7	A character was encountered outside the domain of the collating sequence.
BAD DESCRIPTOR	14	7	The specified conversion descriptor refers to a table that does not exist.
TOO MANY PARAMETERS	15	7	There were too many parameters passed to the service routine.
TOO FEW PARAMETERS	16	7	There were too few parameters passed to the service routine.
OVERLAPPING OPERANDS	17	7	The receiving operand overlaps a part of another operand.
EV NAME INVALID CHAR	20	7	The name of the environment variable contains an invalid character.
EV VALUE INVALID CHAR	21	7	The value of environment variable contains an invalid character.
EV AREA FULL	22	7	The name and value of the specified environment value cannot be stored. The Exec-defined area in the PCT is full. Clear the PCT and restart the run.
EV NAME ILLEGAL FOR TIP	23	7	The name of the environment variable is restricted to the batch and demand execution environment. It is not allowed in the TIP execution environment.
EV NAME NOT FOUND	24	3	The name of the specified environment variable cannot be found.
EV VALUE TOO LONG	25	3	The value of the environment variable is longer than the maximum length allowed in the TIP execution environment.
LOCALE NOT FOUND	26	7	The specified locale cannot be found.
NOT A UNISYS INTERMEDIATE CCS	27	7	The specified locale is not associated with a Unisys intermediate CCS.
INVALID CATEGORY	28	7	The category name specified is not valid.
CATEGORY NOT SET	29	7	The category required by the service routine has not been set.

Table A–1. List of Error Statuses

Message Text	Error Number	Severity Code	Description of Error Status
CORRUPTED TABLE	30	7	The table used by the service routine has been corrupted or an incorrect version of the table is installed..
CCS NOT FOUND	31	7	The specified CCS cannot be found.
FROM CCS EQUALS TO CCS	32	7	In a call to OPEN CCS TO CCS, the specified codes were the same for both the current CCS and the new CCS.
CCS BANK FULL	33	7	The data area containing conversions is full. No more OPEN_CCS_TO_CCS calls can be made.
INVALID BASIC MODE ADDRESS	34	7	The basic mode address could not be converted to an extended mode address.
INVALID EXTENDED MODE ENVIRONMENT	35	7	A problem occurred in trying to establish an extended mode environment for a basic mode caller.
STRING 1 TOO LONG	36	7	The (first) string passed to the service routine is longer than the maximum allowed.
STRING 2 TOO LONG	37	7	The (second) string passed to the service routine is longer than the maximum allowed.
PUT$ENV ERROR	38	7	During a call to PUTENV, the Exec returned an error that is not defined by I18NLIB.
GET$ENV ERROR	39	7	During a call to GETENV, or GETENVSYS, the Exec returned an error that is not defined by I18NLIB.
CCS NUMBER NOT UNIQUE	40	7	Either the CCS SDF number (5) or the CCS ISO number (0) is not unique. Therefore, a CCS name cannot be returned.
ASTORE READ ERROR	41	7	The system was unable to read the subsystem ASTORE word. This word is used to obtain the virtual address (VA) of the data bank that contains category information for a program.
ASTORE WRITE ERROR	42	7	The system was unable to write the virtual address (VA) of a program's category bank to the subsystem ASTORE word.
CREATE BANK ERROR	43	7	The system was unable to create the data bank used to store category information for a program.
CHECKPOINT RESTART ERROR	44	7	After recovering from a system crash, the program restarted at an internal checkpoint, not at the beginning. The system crash invalidated the program's category bank, thereby losing all category data pertinent to that program.

Table A–1. List of Error Statuses

Message Text	Error Number	Severity Code	Description of Error Status
CCS NUMBER FORMAT UNDEFINED	45	7	An invalid format for a CCS number was passed to one of the name-and-number-transformation service routines.
INVALID LOCALE ITEM	50	7	In a call to NL_LANGINFO, an invalid locale item was specified.
INVALID CONVERSION SPEC	51	7	An invalid conversion specification was passed in a format string to either STRFMON, STRFTIME, or STRPTIME.
INVALID ARGUMENT VALUE	52	7	The argument passed to STRFMON does not follow a recognized double-precision floating point number format.
TOO FEW ARGUMENTS (C Compiler only)	53	7	Fewer arguments were passed to STRFMON than are indicated by the format string.
DESCRIPTOR DOES NOT MATCH INPUT CHR	54	7	STRPTIME found an inconsistency while parsing input and format strings.
MODIFIED CONVERSION SPECIFICATION NOT SUPPORTED	55	3	I18NLIB does not support the modified conversion specification used in an STRFTIME or STRPTIME format string.
INVALID DATE/TIME VALUE	56	7	The STRFTIME or STRPTIME call contains an invalid date or time value.
INVALID NMNB PACKET IDENTIFIER	57	7	There is an invalid packet identifier in the call to the NAME_AND_NUMBER packet interface.
INVALID NAME LENGTH IN NMNB PACKET	58	7	The name length in the NAME_AND_NUMBER packet contains an invalid value.
SYSTEM$TIME ERROR	59	7	During a call to SYSTEM$TIME, the Exec returned an error status.

Appendix B
List of Tables

B.1. Abbreviations for Forming Locale Names (ISO 3166:1988)

ISO standard for a coded character set (CCS) with two sets of character code abbreviations (A 2 and A 3) for representing the names of countries. The following is a list of abbreviations used to form locale names:

Table B–1. Abbreviations Used to Form Locale Names

Name of Country	Character Code A 2	Character Code A 3	Code Number
Afghanistan	AF	AFG	004
Albania	AL	ALB	008
Algeria	DZ	DZA	012
American Samoa	AS	ASM	016
Andorra	AD	AND	020
Angola	AO	AGO	024
Anguilla	AI	AIA	660
Antarctica	AQ	ATA	010
Antigua/Barbuda	AG	ATG	028
Argentina	AR	ARG	032
Armenia	AM	ARM	051
Aruba	AW	ABW	533
Australia	AU	AUS	036
Austria	AT	AUT	040
Azerbaijan	AZ	AZE	031
Bahamas	BS	BHS	044
Bahrain	BH	BHR	048
Bangladesh	BD	BGD	050
Barbados	BB	BRB	052

Table B–1. Abbreviations Used to Form Locale Names

Name of Country	Character Code A 2	Character Code A 3	Code Number
Belarus	BY	BLR	112
Belgium	BE	BEL	056
Belize	BZ	BLZ	084
Benin	BJ	BEN	204
Bermuda	BM	BMU	060
Bhutan	BT	BTN	064
Bolivia	BO	BOL	068
Bosnia/Herzegovina	BA	BIH	070
Botswana	BW	BWA	072
Bouvet Island	BV	BVT	074
Brazil	BR	BRA	076
British Indian Ocean Territory	IO	IOT	086
Brunei Darussalam	BN	BRN	096
Bulgaria	BG	BGR	100
Burkina Faso	BF	BFA	854
Burundi	BI	BDI	108
Cambodia	KH	KHM	116
Cameroon	CM	CMR	120
Canada	CA	CAN	124
Cape Verde	CV	CPV	132
Cayman Islands	KY	CYM	136
Central African Republic	CF	CAF	140
Chad	TD	TCD	148
Chile	CL	CHL	152
China	CN	CHN	156
Christmas Island	CX	CXR	162
Cocos (Keeling) Islands	CC	CCK	166
Colombia	CO	COL	170
Comoros	KM	COM	174
Congo	CG	COG	178
Cook Islands	CK	COK	184
Costa Rica	CR	CRI	188

Table B–1. Abbreviations Used to Form Locale Names

Name of Country	Character Code A 2	Character Code A 3	Code Number
Cote D'ivoire	CI	CIV	384
Croatia (Hrvatska)	HR	HRV	191
Cuba	CU	CUB	192
Cyprus	CY	CYP	196
Czech Republic	CZ	CZE	203
Denmark	DK	DNK	208
Djibouti	DJ	DJI	262
Dominica	DM	DMA	212
Dominican Republic	DO	DOM	214
East Timor	TP	TMP	626
Ecuador	EC	ECU	218
Egypt	EG	EGY	818
El Salvador	SV	SLV	222
Equatorial Guinea	GQ	GNQ	226
Eritrea	ER	ERI	232
Estonia	EE	EST	233
Ethiopia	ET	ETH	210
Falkland Islands (Malvinas)	FK	FLK	238
Faroe Islands	FO	FRO	234
Fiji	FJ	FJI	242
Finland	FI	FIN	246
France	FR	FRA	250
France, (metropolitan)	FX	FXX	249
French Guiana	GF	GUF	254
French Polynesia	PF	PY	258
French Southern Territories	TF	ATF	260
Gabon	GA	GAB	266
Gambia	GM	GMB	270
Georgia	GE	GEO	268
Germany	DE	DEU	276
Ghana	GH	GHA	288
Gibraltar	GI	GIB	292

Table B–1. Abbreviations Used to Form Locale Names

Name of Country	Character Code A 2	Character Code A 3	Code Number
Greece	GR	GRC	300
Greenland	GL	GRL	304
Grenada	GD	GRD	308
Guadeloupe	GP	GLP	312
Guam	GU	GUM	316
Guatemala	GT	GTM	320
Guinea	GN	GIN	324
Guinea-Bissau	GW	GNB	624
Guyana	GY	GUY	328
Haiti	HT	HTI	332
Heard/McDonald Islands	HM	HMD	334
Honduras	HN	HND	340
Hong Kong	HK	HKG	344
Hungary	HU	HUN	348
Iceland	IS	ISL	352
India	IN	IND	356
Indonesia	ID	IDN	360
Iran, Islamic Republic of	IR	IRN	364
Iraq	IQ	IRQ	368
Ireland	IE	IRL	372
Israel	IL	ISR	376
Italy	IT	ITA	380
Jamaica	M	JAM	388
Japan	JP	JPN	392
Jordan	JO	JOR	400
Kazakhstan	KZ	KAZ	398
Kenya	KE	KEN	404
Kiribati	KI	KIR	296
Korea, Democratic People's Republic of	KP	PRK	408
Korea, Republic of	KR	KOR	410
Kuwait	KW	KWT	414

Table B–1. Abbreviations Used to Form Locale Names

Name of Country	Character Code A 2	Character Code A 3	Code Number
Kyrgyzstan	KG	KGZ	417
Lao People's Democratic Republic	LA	LAO	418
Latvia	LV	LVA	428
Lebanon	LB	LBN	422
Lesotho	LS	LSO	426
Liberia	LR	LBR	430
Libyan Arab Jamahiriya	LY	LBY	434
Liechtenstein	LI	LIE	438
Lithuania	LT	LTU	440
Luxembourg	LU	LUX	442
Macau	MO	MAC	446
Macedonia, Former Yugoslav Republic of (Provisional)	MK	MKD	807
Madagascar	MG	MDG	450
Malawi	MW	MWI	454
Malaysia	MY	MYS	458
Maldives	MV	MDV	462
Mali	ML	MLI	466
Malta	MT	MLT	470
Marshall Islands	MH	MHL	584
Martinique	MQ	MTQ	474
Mauritania	MR	MRT	478
Mauritius	MU	MUS	480
Mayotte	YT	MYT	175
Mexico	MX	MEX	484
Micronesia, Federated States of	FM	FSM	583
Moldova, Republic of	MD	MDA	498
Monaco	MC	MCO	492
Mongolia	MN	MNG	496
Montserrat	MS	MSR	500
Morocco	MA	MAR	504

Table B–1. Abbreviations Used to Form Locale Names

Name of Country	Character Code A 2	Character Code A 3	Code Number
Mozambique	MZ	MOZ	508
Myanmar	MM	MMR	104
Namibia	NA	NAM	516
Nauru	NR	NRU	520
Nepal	NP	NPL	524
Netherlands	NL	NLD	528
Netherlands Antilles	AN	ANT	530
New Caledonia	NC	NCL	540
New Zealand	NZ	NZL	554
Nicaragua	NI	NIC	558
Niger	NE	NER	562
Nigeria	NG	NGA	566
Niue	NU	NIU	570
Norfolk Island	NF	NFK	574
Northern Mariana Islands	MP	MNP	580
Norway	NO	NOR	578
Oman	OM	OMN	512
Pakistan	PK	PAK	586
Palau	PW	PLW	585
Panama	PA	PAN	591
Papua New Guinea	PG	PNG	598
Paraguay	PY	PRY	600
Peru	PE	PER	604
Philippines	PH	PHL	608
Pitcairn	PN	PCN	612
Poland	PL	POL	616
Portugal	PT	PRT	620
Puerto Rico	PR	PRI	630
Qatar	QA	QAT	634
Reunion	RE	REU	638
Romania	RO	ROM	642
Russian Federation	RU	RUS	643

Table B–1. Abbreviations Used to Form Locale Names

Name of Country	Character Code A 2	Character Code A 3	Code Number
Rwanda	RW	RWA	646
Saint Kitts/Nevis	KN	KNA	659
Saint Lucia	LC	LCA	662
Saint Vincent/The Grenadines	VC	VCT	670
Samoa	WS	WSM	882
San Marino	SM	SMR	674
Sao Tome/Principe	ST	STP	678
Saudi Arabia	SA	SAU	682
Senegal	SN	SEN	686
Seychelles	SC	SYC	690
Sierra Leone	SL	SLE	694
Singapore	SG	SGP	702
Slovakia (Slovak Republic)	SK	SVK	703
Slovenia	SI	SVN	705
Solomon Islands	SB	SLB	090
Somalia	SO	SOM	706
South Africa	ZA	ZAF	710
Spain	ES	ESP	724
Sri Lanka	LK	LKA	144
St. Helena	SH	SHN	654
St. Pierre/Miquelon	PM	SPM	666
Sudan	SD	SDN	736
Suriname	SR	SUR	740
Svalbard/Jan Mayen Islands	SJ	SJM	744
Swaziland	SZ	SWZ	748
Sweden	SE	SWE	752
Switzerland	CH	CHE	756
Syrian Arab Republic	SY	SYR	760
Taiwan, Province of China	TW	TWN	158
Tajikistan	TJ	TJK	762
Tanzania, United Republic of	TZ	TZA	834
Thailand	TH	THA	764

Table B–1. Abbreviations Used to Form Locale Names

Name of Country	Character Code A 2	Character Code A 3	Code Number
Togo	TG	TGO	768
Tokelau	TK	TKL	772
Tonga	TO	TON	776
Trinidad/Tobago	TT	TTO	780
Tunisia	TN	TUN	788
Turkey	TR	TUR	792
Turkmenistan	TM	TKM	795
Turks/Caicos Islands	TC	TCA	796
Tuvalu	TV	TUV	798
Uganda	UG	UGA	800
Ukraine	UA	UKR	804
United Arab Emirates	AE	ARE	784
United Kingdom	GB	GBR	826
United States	US	USA	840
United States (minor outlying islands)	UM	UMI	581
Uruguay	UY	URY	858
Uzbekistan	UZ	UZB	860
Vanuatu	VU	VUT	548
Vatican City State (Holy See)	VA	VAT	336
Venezuela	VE	VEN	862
Viet Nam	VN	VNM	704
Virgin Islands (British)	VG	VGB	092
Virgin Islands (U.S.)	VI	VIR	850
Wallis/Futuna Islands	WF	WL	876
Western Sahara	EH	ESH	732
Yemen	YE	YEM	887
Yugoslavia	YU	YUG	891
Zaire	ZR	ZAR	180
Zambia	ZM	ZMB	894
Zimbabwe	ZW	ZWE	716

B.2. Character Codes for Creating Locale Names (ISO 639:1984)

ISO standard codes for representing of the names of languages. The following are the two-character codes used to create locale names:

aa	Afar
ab	Abkhazian
af	Afrikaans
am	Amharic
ar	Arabic
as	Assamese
ay	Aymara
az	Azerbaijani
ba	Bashkir
be	Byelorussian
bg	Bulgarian
bh	Bihari
bi	Bislama
bn	Bengali
bn	Bangla
bo	Tibetan
br	Breton
ca	Catalan
co	Corsican
cs	Czech
cy	Welsh
da	Danish
de	German
dz	Bhutani
el	Greek
en	English
eo	Esperanto
es	Spanish
et	Estonian
eu	Basque

fa	Persian/Farsi
fi	Finnish
fj	Fiji
fo	Faeroese
fr	French
fy	Frisian
ga	Irish Gaelic
gd	Scots Gaelic
gl	Galician
gn	Guarani
gu	Gujarati
ha	Hausa
hi	Hindi
hr	Croatian
hu	Hungarian
hy	Armenian
ia	Interlingua
ie	Interlingue
ik	Inupiak
in	Indonesian
is	Icelandic
it	Italian
iw	Hebrew
ja	Japanese
ji	Yiddish
jw	Javanese
ka	Georgian
kk	Kazakh
kl	Greenlandic
km	Cambodian
kn	Kannada
ko	Korean
ks	Kashmiri
ku	Kurdish

ky	Kirghiz
la	Latin
ln	Lingala
lo	Laothian
lt	Lithuanian
lv	Latvian
lv	Lettish
mg	Malagasy
mi	Maori
mk	Macedonian
ml	Malayalam
mn	Mongolian
mo	Moldavian
mr	Marathi
ms	Malay
mt	Maltese
my	Burmese
na	Nauru
ne	Nepali
nl	Dutch
no	Norwegian
oc	Occitan
om	(Afan) Oromo
or	Oriya
pa	Punjabi
pl	Polish
ps	Pashto
ps	Pushto
pt	Portuguese
qu	Quechua
rm	Rhaeto-Romance
rn	Kirundi
ro	Romanian
ru	Russian

rw	Kinyarwanda
sa	Sanskrit
sd	Sindhi
sg	Sangro
sh	Serbo-Croatian
si	Singhalese
sk	Slovak
sl	Slovenian
sm	Samoan
sn	Shona
so	Somali
sq	Albanian
sr	Serbian
ss	Siswati
st	Sesotho
su	Sudanese
sv	Swedish
sw	Swahili
ta	Tamil
te	Tegulu
tg	Tajik
th	Thai
ti	Tigrinya
tk	Turkmen
tl	Tagalog
tn	Setswana
to	Tonga
tr	Turkish
ts	Tsonga
tt	Tatar
tw	Twi
uk	Ukrainian
ur	Urdu
uz	Uzbek

vi	Vietnamese
vo	Volapuk
wo	Wolof
xh	Xhosa
yo	Yoruba
zh	Chinese
zu	Zulu

B.3. Locale Names and Numbers Used With the CONVERT_FORM Service Routine

As part of the migration process, the CONVERT_FORM service routine transliterates a character from one representation to another. A locale value is used to determine which transliteration table to use. For more details, refer to the CONVERT_FORM Service Routine topic.

Migration Identifier

An MIG prefix in a locale name indicates that the locale has been adapted for migration purposes.

Currently Available Locales

The following list contains the locale names (column 1) and corresponding numeric identifiers (column 4) that are used by the CONVERT_FORM service routine.

Note: *This list includes only those locales that are supported by the I18NLIB service routines; it does not include all locales that have been developed for migration.*

Locale Name	Language/Where Used	Associated Charmap	Locale Number
da_DK.MIG646DK*	Danish/Denmark	MIG646DK	44
de_DE.MIG646DE*	German/Germany	MIG646DE	47
en_CA.MIG646CA*	English/Canada	MIG646CA	49
es_ES.MIG646ES*	Spanish/Spain	MIG646ES	52
fr_CH.MIG646FR*	French/Switzerland	MIG646FR	56

B.4. Locale Names and Numbers Used With the Locale-Name-and-Number Service Routines

The locale-name service routines are used to convert locale names to their corresponding locale numbers, and the locale-number service routines are used to

convert locale numbers to their corresponding locale or CCS names. For more details, refer to the topics describing these service routines:

- LOCALE_NAME_TO_LOCALE_NUM
- LOCALE_NUM_TO_LOCALE_NAME
- LOCALE_NUM_TO_CCS_NUM

B.5. Modified Conversion Specifications Table (STRFTIME and STRPTIME)

The I18NLIB interprets conversion specifications (CS) with E or O modifiers as if they were unmodified.

This MODIFIED CS	Is interpreted as this UNMODIFIED CS	This MODIFIED CS	Is interpreted as this UNMODIFIED CS	This MODIFIED CS	Is interpreted as this UNMODIFIED CS
%Ec	%c	%Oe	%e	%OU	%U
%EC	%C	%OH	%H	%OV	%V
%Ex	%x	%OI	%I	%Ow	%w
%EX	%X	%Om	%m	%OW	%W
%Ey	%y	%OM	%M	%Oy	%y
%EY	%Y	%OS	%S		
%Od	%d	%Ou	%u		

B.6. National Replacement Character Set (NRC)

A character set in which certain ISO 646 graphic characters (#, $, @, [,], \) are replaced with other characters or symbols to satisfy local language needs. The following are the replacement characters for several ISO 646 variants:

Octal Character	043	044	100	133	134	135	136	140	173	174	175	176
Hexadecimal Character	23	24	40	5B	5C	5D	5E	60	7B	7C	7D	7E
ISO 646 US	#	$	@	[	\	]	^	`	{	|	}	~
ISO 646UK UK	£	$	@	[	\	]	^	`	{	|	}	~
ISO 646FR France	£	$	à	°	ç	§	^	`	é	ù	è	¨

ISO 646DE Germany	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
ISO 646IT Italy	£	$	§	°	#	é	^	ù	à	ò	è	ì
ISO 646SE Sweden	#	¤	É	Ä	Ö	Å	Ü	é	ä	ö	å	ü
ISO 646DK Denmark	#	$	@	Æ	Ø	Å	^	`	æ	ø	å	~
ISO 646NO Norway	#	$	@	Æ	Ø	Å	^	`	æ	ø	å	~

B.7. Standard Transliteration Table

CCS Names and Descriptions

The following coded character sets are supported by the I18NLIB service routines. See Table B–3 for a description of the column names.

Table B–2. Transliteration Table CCS Names and Description

CCS Name	Description	Aliases	ISO CCS #	SDF CCS #	Unisys CCS #
Unisys Character Sets					
Fieldata	Unisys 2200 Fieldata Character Set		0	0	0
BICS	Unisys MAPPER Basic International Character Set		0	47	47
ASCII Character Sets					
ASCII	American Standard Character Set for Information Interchange	ANSI-X3-4; ISO646US; CP367; IBM367; ISO-IR-006; ISO_646.IRV:1991	60	1	1
EBCDIC Suite of Character Sets					
EBCDIC-US	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987		0	3	92
ISO 646 Suite of Character Sets					
ISO646CA	CSA_Z243.4-1985-1	CSA-243-1; CSA7-1; ISO-IR-121	1210	26	26

Table B–2. Transliteration Table CCS Names and Description

CCS Name	Description	Aliases	ISO CCS #	SDF CCS #	Unisys CCS #
ISO646DE	DIN_66003	DIN-66003 ISO-IR-021	210	20	20
ISO646ES	Spanish - Olivetti	ISO-IR-017	170	23	23
ISO646FI	SEN_850200_B	ISO-IR-010; ISO646SE	100	25	25
ISO646FR	NF_Z_62-010 (1982)	ISO-IR-069	690	19	19
ISO646GB	BS_4730	ISO-IR-004	40	21	21
ISO646IT	Italian - Olivetti	ISO-IR-015	150	22	22
ISO646NO	NS_4551-1	ISO-IR-060	600	18	18
ISO646PT	Portuguese - Olivetti	ISO-IR-016	160	28	28
ISO646SE	SEN_850200_B	ISO-IR-010; ISO646FI	100	24	24
ISO646US	ANSI_X3.4-1968	ANSI-X3-4	60	17	188
ISO 8859 Suite of Character Sets					
ISO8859-1	ISO 8859.1:1987	LATIN1	1000	35	35
IOS8859-2	ISO 8859.2:1987	LATIN2	1010	36	36
ISO8859-3	ISO 8859.3:1988	ISO-IR-109; LATIN3	1090	37	37
ISO8859-4	ISO 8859.4:1988	ISO-IR-110; LATIN4	1100	38	38
ISO8859-5	ISO 8859.5:1988	CYRILLIC; ISO-IR-144	1440	39	39
ISO8859-6	ISO 8859.6:1987	ARABIC; ASMO-708; ECMA-114; ISO-IR-127	1270	40	40
ISO8859-7	ISO 8859.7:1987	ECMA-118; ELOT_928; GREEK8; ISO-IR-126	1260	41	41
ISO8859-8	ISO 8859.8:1988	ISO-IR-138	1380	42	42
ISO8859-9	ISO 8859.9:1989	ISO-IR-148; LATIN5	1480	43	43
ISO8859-10	ISO 8859.10:1993	ISO-IR-157; LATIN6	1570	44	44
UCS-2 Character Set					

Table B–2. Transliteration Table CCS Names and Description

CCS Name	Description	Aliases	ISO CCS #	SDF CCS #	Unisys CCS #
UCS-2	UCS-2 form of the ISO10646 CCS		0	61	61
Japanese Character Sets					
JIS-X0201	JIS_X0201 JIS_C6220-1969-JP	ISO-IR-013; KATAKANA; X0201-7; JIS-C6220	0	5	160
LETS-J	Japanese LETSJ Charmap.	NUL-KANJI; LETS/J	0	5	195
Character Set Specifications for RDMS 2200 For details, go to Specifying a Character Set on a Column Definition.					
ASCII	American Standard Character Set for Information Interchange	ANSI-X3-4; ISO646US; CP367; IBM367; ISO-IR-006; ISO_646.IRV:1991	60	1	1
ASCII_FULL	Charmap used by the Relational Database Server to fulfill the requirements of FIPS PUB 127. It is based on ISO 8859-1.	ISO-IR-100; ISO-8859-1; LATIN1; L1; IBM819; CP819	0	5	191
ASCII_GRAPHIC	Charmap used by the Relational Database Server to fulfill the requirements of FIPS PUB 127. It is based on ISO 8859-1.	ISO-IR-100; ISO-8859-1; LATIN1; L1; IBM819; CP819	0	5	190
LATIN1	ISO 8859.1:1987	ISO8859-1	0	5	192
RDMS_TEXT	Charmap used by the Relational Database Server to provide migration for existing data.	ISO-8859-1; LATIN1; L1; IBM819; CP819	0	5	189
SQL_CHARACTER	Charmap used by the Relational Database Server to fulfill the requirements of FIPS PUB 127. It is based on ISO 8859-1.	ISO-IR-100; ISO-8859-1; LATIN1; L1; IBM819; CP819	0	5	193

Table B–2. Transliteration Table CCS Names and Description

CCS Name	Description	Aliases	ISO CCS #	SDF CCS #	Unisys CCS #
SQL_TEXT	Charmap used by the Relational Database Server to fulfill requirements of SQL 92.	ISO-IR-100; ISO-8859-1; LATIN1; L1; IBM819; CP819	0	5	194
Unisys Migration Character Sets For details, go to Locale Names and Numbers Used With the CONVERT_FORM Service Routine.					
MIG646CA	Unisys Migration CCS for ISO 646 Canada		0	5	176
MIG646DE	Unisys Migration CCS for ISO 646 Germany		0	5	177
MIG646DK	Unisys Migration CCS for ISO 646 Denmark		0	5	178
MIG646ES	Unisys Migration CCS for ISO 646 Spain		0	5	179
MIG646FR	Unisys Migration CCS for ISO 646 France		0	5	181

Column Names and Descriptions

Table B–3. Transliteration Table Column Names and Description

Name of Column	Column Definition	Used for	Returned as
CCS Name	The name for the CCS definition. This name is assigned to the CCS by your system administrator when the transliteration table is installed.	The ccs_name parameter on the CCS NAME TO CCS NUM service routine call. The tocode and fromcode parameter values of the C Compiler iconv_open() function.	The name in the ccs_name parameter on the CCS NUM TO CCS NAME service routine call.
Description	The descriptive name for the CCS.	Documentation purposes only	N/A
Aliases	Other common names for the CCS.	Documentation purposes only	N/A

Table B–3. Transliteration Table Column Names and Description

Name of Column	Column Definition	Used for	Returned as
ISO CCS #	The decimal numeric identifier for the CCS from the ISO CCS registry. The value is 10 times the actual registry number. If the value is 0, no number is assigned.	Interpreting or constructing file or message images encoded in the ISO 2022 encoding scheme. Setting the ccs-number-format parameter of ccs-number to iso-ccs-number. The ccs_number parameter of the CCS NUM TO CCS NAME service routine.	The ccs-number parameter of the CCS NAME TO CCS NUM service routine call when you set ccs-number-format to iso-ccs-number.
SDF CCS #	The octal numeric identifier for the CCS. This number is always in the 0-63 range (00-77 octal). This number can appear in the character set field in a system data format (SDF) control image. The value of 5 means that no SDF-unique CCS number is assigned to this CCS.	The ccs-number parameter of the CCS NUM TO CCS NAME service routine when you set the ccs-number-format parameter to sdf-ccs-number. (For more information on using SDF CCS numbers in SDF files, refer to the SDF character set type	The ccs-number parameter of the CCS NAME TO CCS NUM service routine call when you set ccs-number-format to sdf-ccs-number.
Unisys CCS #	The decimal identifier for the Unisys-assigned CCS.	The from-ccs-to-ccs parameter on the OPEN CCS TO CCS service routine call. The ccs-number parameter of the CCS NUM TO CCS NAME service routine when you set ccs-number-format to ccs-number.	The ccs-number parameter of the CCS NAME TO CCS NUM service routine call when you set ccs-number-format to ccs-number.

© 2009 Unisys Corporation.
All rights reserved.

7850 5393–006